

ELECTRICAL

E
N
G
-
N
E
E
R
I
N
G

PRICES SUBJECT TO CHANGE

ENGINEERING EXPERIMENT STATION

AUBURN UNIVERSITY

AUBURN, ALABAMA

(NASA-CR-120487) A COMPARISON OF
SPECIAL-PURPOSE AND GENERAL-PURPOSE
COMPUTERS FOR DATA COMPRESSION Interim
Report, 1 Sep. 1972 - Aug. 1973 (Auburn
Univ.) 85 p

N74-77381

00/99

Unclas
03538

A COMPARISON OF SPECIAL-PURPOSE AND
GENERAL-PURPOSE COMPUTERS
FOR DATA COMPRESSION

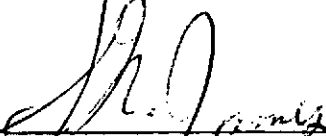
Prepared by

TELEMETRY SYSTEMS LABORATORY
AUBURN UNIVERSITY
AUBURN, ALABAMA

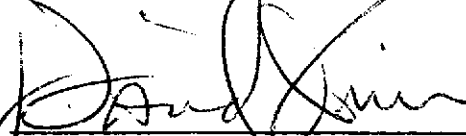
RESEARCH PERSONNEL
H. C. COBB, IV AND S. N. JAMES
INTERIM REPORT
November 1, 1973

CONTRACT NAS8-20765
GEORGE C. MARSHALL SPACE FLIGHT CENTER
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
HUNTSVILLE, ALABAMA

SUBMITTED BY:


S. N. James
Assistant Professor
Co-Project Leader

APPROVED BY:


J. David Irwin
Associate Professor and Head
Electrical Engineering

Foreword

This technical report has been prepared by the Auburn University Telemetry Systems Laboratory, during the period from September 1, 1972 to August 1973, in partial fulfillment of NASA contract NAS8-20765.

Table of Contents

I.	Introduction	1-1
II.	Data Compression Algorithms.	2-1
	A. General Information.	2-1
	B. The Zero Order Interpolator.	2-3
III.	System Description	3-1
	A. System Inputs.	3-1
	B. Time Divisions	3-2
	C. Sampling Rates	3-3
	D. Data Output.	3-3
	E. Nominal Tolerance, Channel Priority, and Buffer Queue	3-5
	F. Confidence Sampling.	3-8
	G. Determination of T_m	3-9
	H. Miscellaneous Preprogrammed Items.	3-10
	I. Operating Procedure.	3-11
IV.	Data Compression Implementation: The Special-purpose Computer	4-1
	A. General Information.	4-1
	B. Speed.	4-2

C.	Hardware Requirements.4-10
D.	Cost4-12
E.	The TDC-1 Special-Purpose Data Compression Computer4-15
V.	Data Compression Implementation: The General-Purpose Computer	5-1
A.	General Information.	5-1
B.	Main Memory Media.	5-2
C.	Speed.	5-7
D.	Programming Techniques	5-9
E.	The Critical Path.	5-13
F.	System Operation Flowchart	5-14
G.	The IBM 4 π Model CP General-purpose Digital Computer	5-29
H.	Multi-processor General-purpose Computers.	5-33
I.	Costs.	5-35
VI.	Conclusion	6-1
	References	R-1

I. Introduction

The requirement to transmit ever increasing amounts of data between various points has, over the years, posed at least one major problem--- how to economically expand data transmitting and receiving facilities in order to provide the necessary data handling capabilities to meet the requirement. In this light, space vehicle telemetry systems have recently undergone considerable investigation of the problem. The most direct approach, of course, is to increase the communications link capacity. However, in many applications, particularly space vehicle telemetry systems, the cost of this approach very rapidly becomes prohibitive. A less direct but more economical approach is the use of data compression, which can reduce bandwidth and/or time requirements so that a greater volume of data can be transmitted over existing facilities.

Implementation of most of the data compression algorithms of interest dictates the need for a system which realizes three basic functions--- memory, control, and arithmetic. Since each of these functions is inherent in a digital computer, the question of implementation often reduces to the selection of special purpose (SP) or general purpose (GP) hardware and the associated software.

The purpose of this report is to present a study of a special case, namely, implementation of data compression aboard the proposed space shuttle and to draw conclusions as to which type of digital computer (SP or GP) would be best for this application.

II. Data Compression Algorithms

A. General Information

There is quite a large number of data compression techniques, or algorithms, available. Usually they are referred to as "redundancy removal" algorithms, being classified according to the order of the algorithm. Redundancy exists when there is an insignificant change between successive data samples, or among several samples. In order to determine sample redundancy, it is common practice to monitor a particular derivative, since a derivative is generally associated with change. The order of the derivative chosen thus determines the order of the particular algorithm.

For space telemetry systems, consideration is generally given to four algorithms---zero-order predictor, zero-order interpolator, first-order predictor, and first-order interpolator. Although this report is not primarily concerned with the analysis of the various data compression algorithms available, a few descriptive remarks are in order.

In general, the higher the order of the algorithm the more complicated it becomes. It might also be noted that interpolators are more complicated than predictors. However, all algorithms are similar in that they use one or more previous data points to establish a reference "corridor" inside of which future data samples are expected to lie. If it is determined that a subsequent sample falls outside of the corridor boundaries, then this sample is said to be non-redundant, and it is used, alone or with other samples, to determine the boundaries of a new reference corridor. It is this non-redundant sample that carries the most

useful information, therefore, a system is desired that will analyze input data samples and reject or discard redundant samples, while transmitting all non-redundant samples.

Naturally, when the waveform is reconstructed there is going to be some error due to the elimination of redundant data. This error will, quite obviously, vary with both the input data or waveform and the type of compression algorithm employed. The RMS error is quite often used as one of the major performance criteria.

Another major criterion is the compression ratio which a particular algorithm realizes. The compression ratio is defined by the equation

$$CR = \frac{S_1}{S_0}$$

where CR is the compression ratio, S_1 is the input sample rate, and S_0 is the average output sample rate. This definition makes CR proportional to the sampling rate, hence two tests which differ only in sampling rate might well yield conflicting compression ratios. Further complicating matters is the fact that the number of non-redundant samples for some algorithms are more dependent on sampling rates than for others. For instance, given two algorithms, one might offer a considerably better CR at one sampling rate, while at a different rate, the situation might be completely reversed. In general, zero-order algorithms are more sensitive to sampling rates than first order algorithms. [9]

In addition to those just mentioned, there are other characteristics of compression algorithms which might be considered in order to have a

clearer understanding of the advantages and disadvantages of various methods. These characteristics are listed below. [9]

- (1) An interpolation algorithm typically gives better results than a corresponding prediction algorithm of the same order.
- (2) First and higher-order algorithms often exhibit oscillatory tendencies which detract seriously from compression efficiency.
- (3) For most data signals, the first order predictor (FOP) and its variations suffer most from oscillatory tendencies.
- (4) Zero-order algorithms, because they suffer least from oscillatory tendencies, quite often provide a compression ratio equal to or greater than the first-order algorithms.

B. The Zero Order Interpolator

As previously mentioned the compression ratio realized by a particular algorithm may vary with the type of input data. Knowing the type of input data, a designer could then implement the best algorithm for compressing the data. Although present knowledge of the data characteristics to be handled on board the space shuttle is quite limited, a study of nearly 900 measurements from four flights of the Saturn vehicle determined that the zero-order interpolator (ZOI) algorithm gives the best performance, that is, the highest compression ratio and the lowest RMS error, of the four algorithms mentioned at the beginning of this chapter. [1] Hence it seems that the ZOI is the best initial choice for the space shuttle.

The general procedure for the ZOI algorithm, sometimes referred to in literature as the zero-order, variable corridor, artificial preceeding sample retained (ZVA) algorithm, is outlined below. [2,10]

- a) The occurrence of a non-redundant sample requires that a new corridor be established. This is done by drawing lines of zero slope through the end points of the tolerance range placed about the non-redundant sample.
- b) For a subsequent sample to be redundant, one end of the tolerance range placed about the sample must fall within the corridor. Each redundant sample modifies the corridor extended to the next sample in the following manner. The new corridor consists of that part of the previous corridor which is overlapped by the tolerance range placed about the redundant sample.
- c) If the tolerance range placed about the sample does not overlap the corridor, the sample is non-redundant. However, it is not retained. Instead, the midpoint of the corridor used to analyze the sample in question, actually a predicted value, is retained for the preceeding sample. Hence the retained sample is an "artificial" or "adjusted" sample rather than a "real-data" sample.

Figure 2-1 illustrates how the ZOI might operate on various data inputs in accordance with the procedure just described. It might be noted

while observing Figure 2-1 that although a sample might be redundant, it need not lie within the corridor.

Although it has been determined that the ZOI seems to have the edge, performance wise, over other algorithms, it will be shown in Chapter IV that the ZOI suffers a disadvantage in terms of hardware implementation. Although it will not be shown in Chapter V, the ZOI algorithm would, due to increased complexity, require more software to implement the general-purpose computer. These disadvantages, although worthy of consideration, are, nonetheless, outweighed by the performance gains realized with the ZOI.

III. System Description

Whether the machine chosen to implement the desired data compression is a general purpose or a special purpose computer, there are certain items which must be observed in order that the machine might meet the basic requirements. These items are the subject of the following discussion.

A. System Inputs [3]

The input data will be in the form of ten-bit data words which may be received from one of sixty-four (64) data channels. For the purpose of this report, these channels will be assumed to be simply commutated. A "data ready" pulse will signal the machine that a new data sample is ready for analysis. Since one of the main concerns in this report is that of maximum word rates, the following definition will be made.

Definition 3-1. The rate at which "data ready" pulses arrive at the input will be denoted by the letter W. Hence W is the rate at which the machine must be able to examine the input data words.

From the above definition it is quite easy to see that once the data ready pulse is input to the computer, the corresponding ten-bit data sample must be accepted within a period of $1/W$.

B. Time Divisions

In order to effectively synchronize the data for subsequent reconstruction, some type of time division must be employed. This will allow insertion of a synchronization word periodically which, in turn, will relate the data to a particular point in time. The following definitions should serve to clarify what is meant by time divisions.

Definition 3-2. Frame: a continuous (in time) block of input data which contains 2048 ten-bit data words (starting point arbitrary). Frame count will be recorded by an index called the main frame count (MFC).

Definition 3-3. Subframe: a division of the frame which contains 64 ten-bit data words. Hence, there are 32 sub-frames per frame. Sub-frame count will be recorded by an index called the subframe number (SFN).

Definition 3-4. Subframe time slot (SFTS): a slot or position within the subframe which is capable of storing one ten-bit data word. Since there are 64 words per sub-frame, there are, therefore, 64 sub-frame time slots per subframe.

One other item should be brought to light with regard to definition 3-4, and that is the fact that since there are 64 data input channels and 64 SFTS's, each SFTS may also be thought of as a channel number or address.

C. Sampling Rates [3]

The maximum rate at which any one channel will be sampled will be denoted by the symbol R_f . Hence if each channel is to be sampled at a rate R_f , then

$$W = 64 R_f,$$

which simply states that the computer must handle input data words at a rate 64 times that of the sample rate. In this report, 6 sampling rates will be considered. These are R_f , $R_f/2$, $R_f/4$, $R_f/8$, $R_f/16$, and $R_f/32$. It may also be desirable to vary the sampling rate between channels, but it will be shown later in Chapter V that this can create idle CPU time in the GP computer.

D. Data Output [3, 9]

By its very nature, a data compressor will accept input data which is regularly spaced in time and output data whose time spacing is quite random. This is undesirable since PCM telemetry systems require constant output data rates. [1] An "uncontrolled output" data compression system is shown in Figure 3-1.

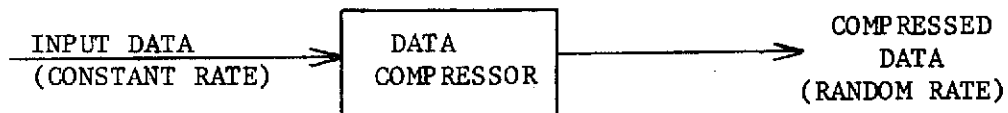


Figure 3-1 Block diagram of uncontrolled output data compressor

The problem of random output data rates may be overcome by the addition of a buffer to the output of the data compressor. Unfortunately, addition of the buffer introduces two major problems:

1) During periods of high data activity the buffer will tend to "overflow", resulting in indiscriminate loss of data. 2) During periods of low data activity the buffer may become completely empty or "underflow", once again resulting in a non-uniform output data rate. Such a system is shown in figure 3-2.

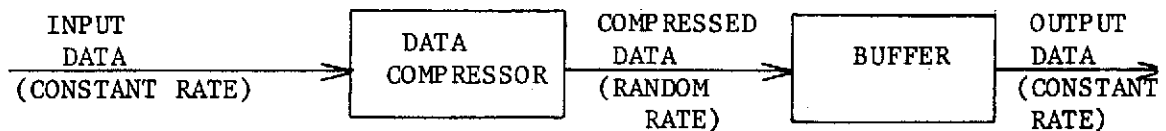


Figure 3-2. Data compression system with uncontrolled buffer

A third, but somewhat less severe problem introduced by the buffer is that of time delay. It is quite obvious that the time delay imposed upon compressed data is proportional to buffer length. Thus the maximum allowable time delay establishes a top limit on buffer length (cost and size may also be limiting factors). Furthermore, there will be periods when the average output rate of the data compressor suddenly becomes quite high (these are referred to as "active" periods during which there is little redundancy in input data). The buffer should therefore, be large enough so as not to easily overflow during these surges of non-redundant data. It might be mentioned, in passing, that any increase in buffer length also tends to ease the buffer queue control problem which will be discussed in a subsequent section. So long as the buffer size is not too large the time delay involved should have little effect on the significance of compressable data. Non-compressable or

critical data (e.g. data which might monitor some catastrophic failure) is, of course, transmitted with no compression and no delay.

Given the system of Figure 3-2, it is reasonable to assume that if information on buffer occupancy could be fed back to the data compressor, the data compressor could, in turn, alter its average output rate so as to prevent buffer overflow or underflow. A buffer controlled or "queue length" data compressor system is shown in Figure 3-3.

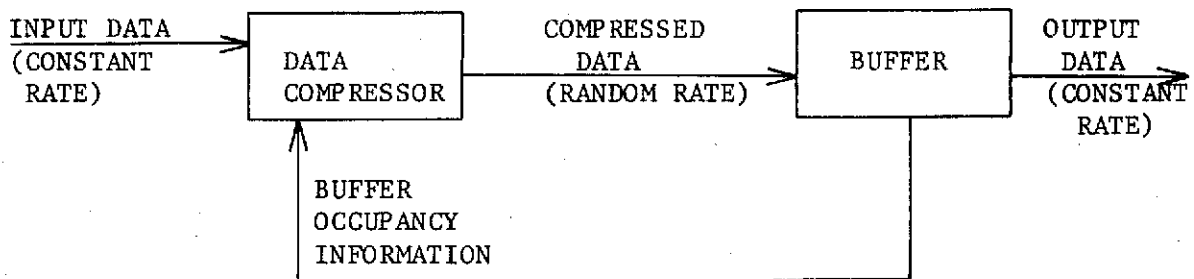


Figure 3-3. A buffer controlled (queue length) data compression system

It seems at present that a buffer length of about 1024 words should keep any time delay error to a minimum while allowing sufficient room for the onrush of data during active data periods.

E. Nominal Tolerance, Channel Priority, and Buffer Queue [3]

In chapter II, the concept of data compression was introduced and it was shown how a tolerance range is used, in conjunction with one or more previous data points, to establish a reference corridor so that the redundancy status of subsequent samples may be determined. The initial tolerance is termed, nominal tolerance, denoted T_n , and it is

desired that the nominal tolerance be easily programmable (i.e. ability to read in values from paper tape, magnetic tape, keyboard, cards, etc.) at the beginning of a given mission. Five values of T_n are to be considered. These are:

- 1) $T_n = \pm 1$
- 2) $T_n = \pm 2$
- 3) $T_n = \pm 4$
- 4) $T_n = \pm 8$
- 5) $T_n = \pm 16$

Since it might be desirable to vary T_n between channels, a provision must be made to store a value of T_n for each channel.

Among other items which should be easily programmable for a given mission are the channel priority, P (a bi-valued variable), and six buffer queue control points, q_1, q_2, \dots, q_6 . A channel may have either high or low priority, depending upon the importance of the data handled by that channel. The buffer queue control points should be selected so as to best prevent the occurrence of buffer overflow or underflow. Throughout the remainder of this report, the following items should be remembered about the buffer queue control points:

- 1) $q_1 > 0$
- 2) $q_6 < 1024$
- 3) $q_1 < q_2 < \dots < q_6$.

Items 1) and 2) are quite obvious, since they deal with the minimum and maximum possible levels of buffer occupancy, respectively.

Although item 3) is merely a convention which states that the value of the control points increases with increasing subscript, it might serve to eliminate any confusion in later discussion.

In section D of this chapter, it was suggested that it might be possible to use feedback information concerning buffer occupancy status to cause the data compressor to alter its average output rate, thereby preventing buffer overflow or underflow. One method used to alter the average output rate of the data compressor is to alter the tolerance used in redundancy analysis. For instance, if the tolerance is increased, the average output of the compressor will decrease. Similarly, a decrease in tolerance will result in an increase in the average compressor output rate. After studying Chapter II, this should be quite intuitive, since an increase in tolerance will increase redundancy (decrease the number of retained or non-redundant samples), while the reverse is true for a decrease in tolerance.

It is convenient at this point to introduce another variable called the altered or modified tolerance, T_m . As will become quite obvious later, it is the modified tolerance that is used in redundancy analysis, however, it should be remembered that under some conditions T_m and T_n are equal. In a subsequent section it will be shown that the exact value which T_m assumes depends upon several factors, among which is T_n .

In the light of the previous discussion it is quite easy to see how buffer occupancy may be controlled. For example, if the buffer becomes loaded such that the occupancy level exceeds some buffer queue point, the modified tolerance will be increased, which in turn lowers

the average compressor output rate, allowing buffer occupancy to decrease. On the other hand, if buffer occupancy becomes dangerously low, the modified tolerance will be decreased, which will result in a tendency to increase buffer occupancy.

F. Confidence Sampling

During periods of low data activity (i.e. periods in which samples vary only a very slight amount), a decrease in the modified tolerance may not always insure that buffer underflow will not occur. When this uncertainty exists, it becomes necessary to retain redundant data samples in order to raise the buffer occupancy level. There are procedures for selecting redundant data points during these periods of low data activity, all of which are called confidence sampling.

One method of confidence sampling calls for the retention of a sample periodically from each channel. This is accomplished by counting the number of rejected or redundant samples succeeding the last retained sample. If this count reaches some predetermined level before another non-redundant sample is encountered, then the sample being tested, when the count reaches this particular level, is retained.

There is another method of confidence sampling which is, in part, controlled by the buffer status. To illustrate, suppose that it is necessary to monitor the data on a particular channel during periods of low activity and further, that the number of data points retained by the periodic method is insufficient for accurate reconstruction. The channel in question could be "tagged" with a confidence sampling bit, and during periods when the buffer occupancy is below a specified level,

every sample from this channel (governed only by the sampling rate) would be retained.

It might be noted that confidence sampling not only helps reduce the buffer underflow problem, but also helps reduce reconstruction problems by eliminating what is called transmission "dropout". Dropout is simply an interruption that occurs between sampling runs or when a channel temporarily becomes completely inactive. The machine which reconstructs the data is unable to determine whether an inactive data period occurred during a run or a run was terminated and a new run was initiated, until new data is received. If this uncertainty exists for more than a very brief period of time, interpretation of the data may become quite difficult.

G. Determination of T_m [3]

The information used in determining the value of T_m includes buffer occupancy status, T_n , and channel priority. Table 3-1 shows how the value of T_m will be determined based upon these factors, and it also shows when confidence sampling will be used. The present buffer status will be denoted by the letter q .

Buffer Status	Procedure
$q < q_1$	Retain all samples
$q_1 < q < q_2$	$T_m = T_n$ and confidence sampling
$q_2 < q < q_3$	$T_m = T_n$ on all channels
$q_3 < q < q_4$	$T_m = 2T_n$ on all channels
$q_4 < q < q_5$	$T_m = 2T_n$ on high priority channels and $T_m = 4T_n$ on low priority channels
$q_5 < q < q_6$	$T_m = 4T_n$ on high priority channels and delete low priority channels
$q > q_6$	Delete all channels

Table 3-1. Values of T_m for all possible values of buffer occupancy status (q).

H. Miscellaneous Preprogrammed Items [3]

In addition to those previously discussed, there are three other items which must be preprogrammed before a mission. These items are:

- 1) A sixteen-bit synchronization word.
- 2) The buffer occupancy information control word (BOIC) which is an eleven bit word. The five most significant bits of this word indicate the SFN during which the buffer occupancy information will be transmitted, while the six least significant bits indicate the SFTS or channel over which this information will be transmitted.
- 3) The compressed data output rate, R , which specifies, in terms of "data-ready" pulses, the rate at which words are output from the buffer. An index called the output synchronization

count (OSC) will count the data ready pulses. When the OSC becomes equal to R, a word is output from the buffer and the OSC is reset.

I. Operating Procedure [3]

Having laid the groundwork in the previous sections, the general operating procedure, by which the system must abide, may be introduced.

- 1) The arrival of a data-ready pulse signals that a new sample is ready for examination. This sample must be accepted within a period of $1/W$.
- 2) Increment the SFTS index. If the SFTS index becomes greater than 63, reset it and increment the SFN. The SFN resets when it tries to exceed 31.
- 3) Using the SFTS index, access the memory location which contains the sampling rate for this SFTS (channel).
- 4) If the rejected data count (RDC) for this channel is equal to 31, reset the RDC and force acceptance of this sample; proceed to step 8.
- 5) Obtain the existing reference corridor for this channel.
- 6) Determine value of T_m , based on buffer occupancy status, priority, confidence sampling tag, and T_n (see Table 3-1).
- 7) Make a decision to retain or discard the sample. Determine the new reference corridor values and return them to memory.
- 8) If the sample is retained combine with the SFTS index (to form a sixteen-bit word) and put the combined information

into the buffer.

- 9) After each 64 retained samples, insert the 16-bit synchronization identification word or its complement (the synch word is alternated with its complement).
- 10) When the MFC resets (it is incremented each time the SFN is reset, and is reset when it tries to exceed 16), insert into the buffer, at a time dictated by the buffer occupancy information control word (BOIC), a ten-bit reading of buffer occupancy. This reading will be combined with the channel address over which the information is carried (also dictated by the BOIC word).
- 11) Once each R data ready pulses, output a 16 bit word from the buffer.

Of course, hardware or software implementation of this procedure would be a bit more complicated than the procedure actually appears to be. This presentation is merely to show the type of general procedure that the data compression system, whether implemented by a general purpose or a special purpose computer, must observe.

IV. Data Compression Implementation: The Special-purpose Computer

A. General Information

In Chapter III, the desired functions of the data compression system were introduced. Discussion in this chapter will, therefore, turn toward data compression by a particular class of machine, namely, the special purpose (SP) computer.

The special-purpose computer, as the name might imply, possesses rather limited capabilities in terms of versatility of application. For example, a special-purpose computer may be able to perform any one of several data compression algorithms, however, its ultimate goal is data compression and data compression alone. The machine may not easily be modified (in fact, it would be easier to build another machine) or "reprogrammed" to perform, say, guidance functions. It is interesting to note at this point that there is yet some confusion over the exact use of the terms "special-purpose" and "general purpose". Consider, for example, the special-purpose computer which is capable of executing one of several data compression algorithms, depending on the position of one or more control panel switches. This machine is very often referred to as a "general-purpose data compression computer", while a single-algorithm machine is termed a "special-purpose data compression computer". Although the multi-algorithm machine indeed possesses more computational versatility than the single-algorithm machine, it nonetheless has only one ultimate function---data compression. In order to

avoid any further confusion on this matter, the following definition will be made.

Definition 4-1 A special purpose (SP) computer is a machine which possesses a single ultimate function and, depending on design, may realize this function by one or more computational procedures.

From definition 4-1, it is quite easy to see that both the single-algorithm and multi-algorithm data compression computers fall under the classification of special-purpose computers.

B. Speed

Even though the versatility of SP computers is quite limited, the factors which act to limit versatility greatly enhance speed capabilities. One of the main factors involved in the increase in speed is the reduction of instruction acquisition time, or the time required for the machine to obtain instructions from memory. In a special-purpose machine, instructions are usually permanently stored by a method called "hard wiring", i.e. using combinational logic circuits to decode some easily obtainable parameter such as a clock pulse count (actually, as it will be shown later, a counter is sometimes used which counts every K^{th} clock pulse). The counter or register in which this count is recorded is often termed the program counter, while the count itself is usually called the instruction address. Thus the only time delay involved is the propagation delay between the time that the instruction address is available at the input of the logic circuitry and the time that the decoded information,

or instruction is available at the output. Depending on the type of logic circuitry employed and other design factors, this propagation delay may range anywhere from about 50 nsec to about 400 nsec. In designing the decoding logic, or address decoder, one must also take into consideration the fact that conditional transfers or jumps may be required during instruction execution and, therefore, provisions should be made so that the processor might be able to alter the instruction address. In this manner, the proper instruction sequence will be maintained. A block diagram of such a system is shown in Figure 4-1. The typical operation of this type system is the subject of the following discussion.

Initial assumptions: 1) The clock is some type of "free running"

timing device such as an astable multivibrator, a crystal controlled oscillator, etc.

2) The divide by K logic is positive edge-triggered and provides an output pulse every K clock pulses.

3) The system is ready to begin analysis of the next data sample (the program counter is set at the address of the last instruction executed and the divide by K circuit is reset and inhibited).

Typical operation: 1) Upon receiving a data ready pulse, the processor first sets the program counter to the location of the first instruction in the sequence. After a time just slightly longer (by a few nanoseconds) than the propagation delay time of the address decoder, an "instruction ready" pulse will signal the processor that the delay time has expired. At this time the processor sets the divide

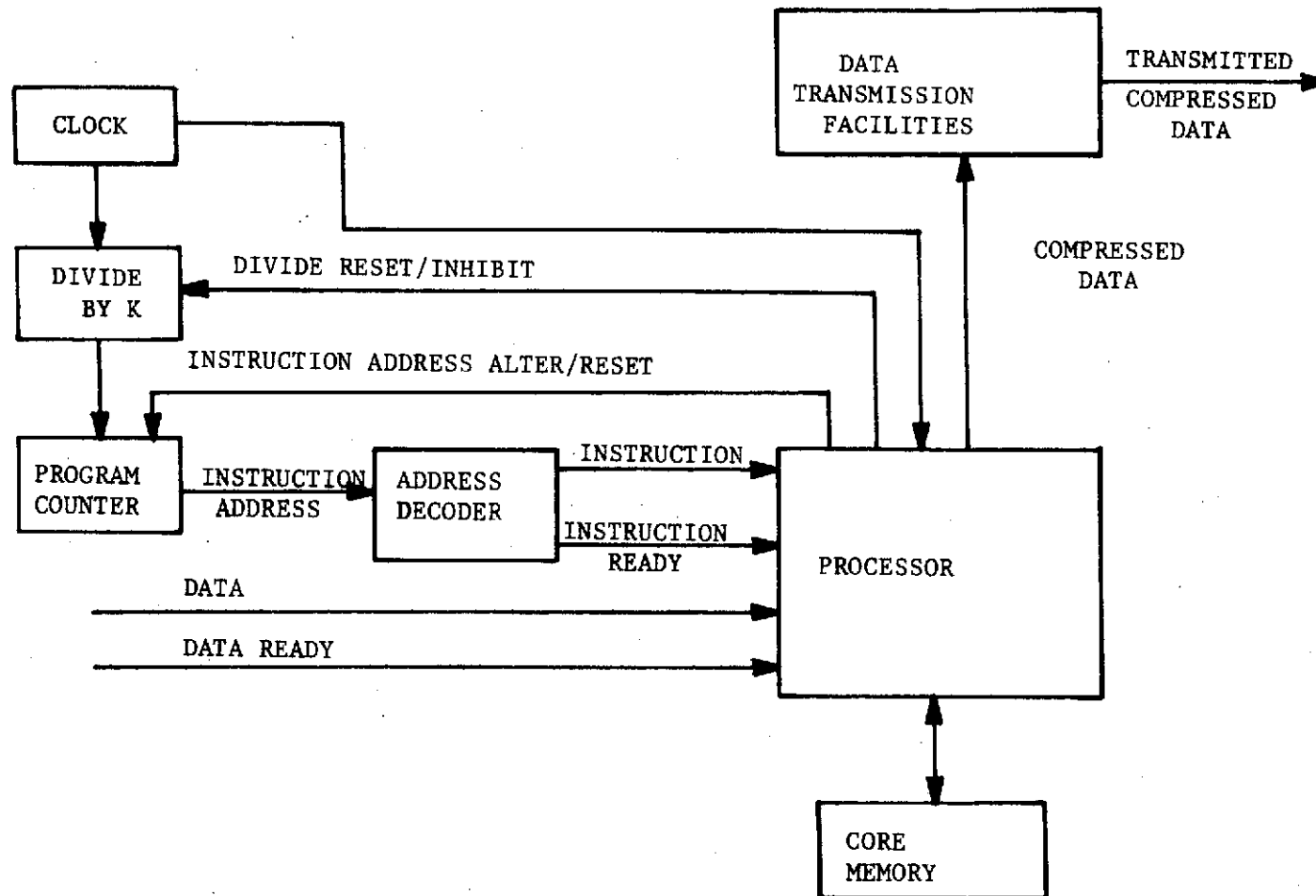


Figure 4-1. Block diagram of a special-purpose data compression computer which uses a hard-wired address decoder for instruction acquisition.

reset/inhibit line to zero and begins to execute the first instruction.

- 2) On the K^{th} clock pulse the divide by K circuit provides a pulse which increments the program counter (as will be discussed shortly, the processor has now had sufficient time to complete execution of the first instruction and is now in a "wait" mode). When the "instruction ready" pulse arrives, the processor will begin to execute the next instruction.
- 3) When a transfer or jump is required, the processor sets the divide reset/inhibit line to one, which resets and halts the divider. Next, the processor sets the program counter to the proper count or address to obtain the next instruction. When the "instruction ready" pulse arrives the processor sets the divide reset/inhibit line to zero (allowing the divide by K circuit to resume operation) and begins to execute the next instruction.
- 4) At the end of the instruction sequence, the processor sets the divide reset/inhibit line to one and enters a "wait" mode until a new data sample becomes available. Upon the arrival of a new data sample, steps 1) through 4) are repeated.

The instruction acquisition time for a system such as the one of Figure 4-1 can be quite small. For instance, the delay between the time an instruction is completed and the time the next instruction may begin execution might range, depending on the design and hardware used, from

about 75 nsec to about 425 nsec. Considering the present state of integrated circuit technology, it would be reasonable to assume that an instruction acquisition time of about 200 nsec could be realized with no great amount of trouble or expense. However, there exists potential for even greater speed, which may be realized with only slight modifications to the system of Figure 4-1.

Consider once again the system just discussed. In selecting the minimum allowable value for K, one must decide how many clock pulses are required for the processor to execute the longest (that is the most time consuming) instruction in the data compression algorithm. This number of clock pulses is the minimum value which may be assigned to K and still maintain proper system operation. It is immediately obvious, however, that this arrangement will almost surely consume more time than is actually needed. To illustrate, consider an algorithm composed of a set of instructions which may be broadly classified into three categories---long, medium, and short. Further, suppose that the medium instruction requires half as long to execute as the long instruction and that the execution time of the short instruction is one fourth that of the long instruction. The execution times for the long, medium, and short instructions will be denoted t_l , t_m , and t_s respectively. If the algorithm contains N_l long instructions, N_m medium instructions and N_s short instructions, then the total time required (T) for the system in Figure 4-1 to process all the instructions is given by the equation

$$T = t_l (N_l + N_m + N_s).$$

On the other hand, if a system used only the amount of time required by a given instruction, the total time required (T) would be given by the equation

$$T = N_{\ell}t_{\ell} + N_m t_m + N_s t_s.$$

Thus, a system of this type uses an amount of time,

$$\Delta T = N_m(t_{\ell}-t_m) + N_s(t_{\ell}-t_s), \text{ less than the system of Figure 4-1.}$$

As a numerical example, consider the following parameters:

$$N_{\ell} = 25, N_m = 55, N_s = 20, \text{ and } t_{\ell} = 4\mu\text{sec.}$$

For the system of Figure 4-1,

$T = 4(25 + 55 + 20)\mu\text{sec} = 400\mu\text{sec}$, whereas a system using only the time required for each instruction would use an amount,

$T = 4(25)\mu\text{sec} + 55(2)\mu\text{sec} + 20(1)\mu\text{sec}$, or $T = 230 \mu\text{sec}$, and $\Delta T = 170 \mu\text{sec}$, which represents a 42.5% reduction in execution time with the values given.

Systems which use only the amount of time required for an instruction to execute are sometimes called "completion countdown" systems, and a provision is usually made so that the processor will signal upon completing execution of each instruction. The term "completion countdown" comes from the fact that in some processors the address decoder not only provides the instruction, but also provides the processor with the time requirement (i.e. the number of clock pulses required) for that instruc-

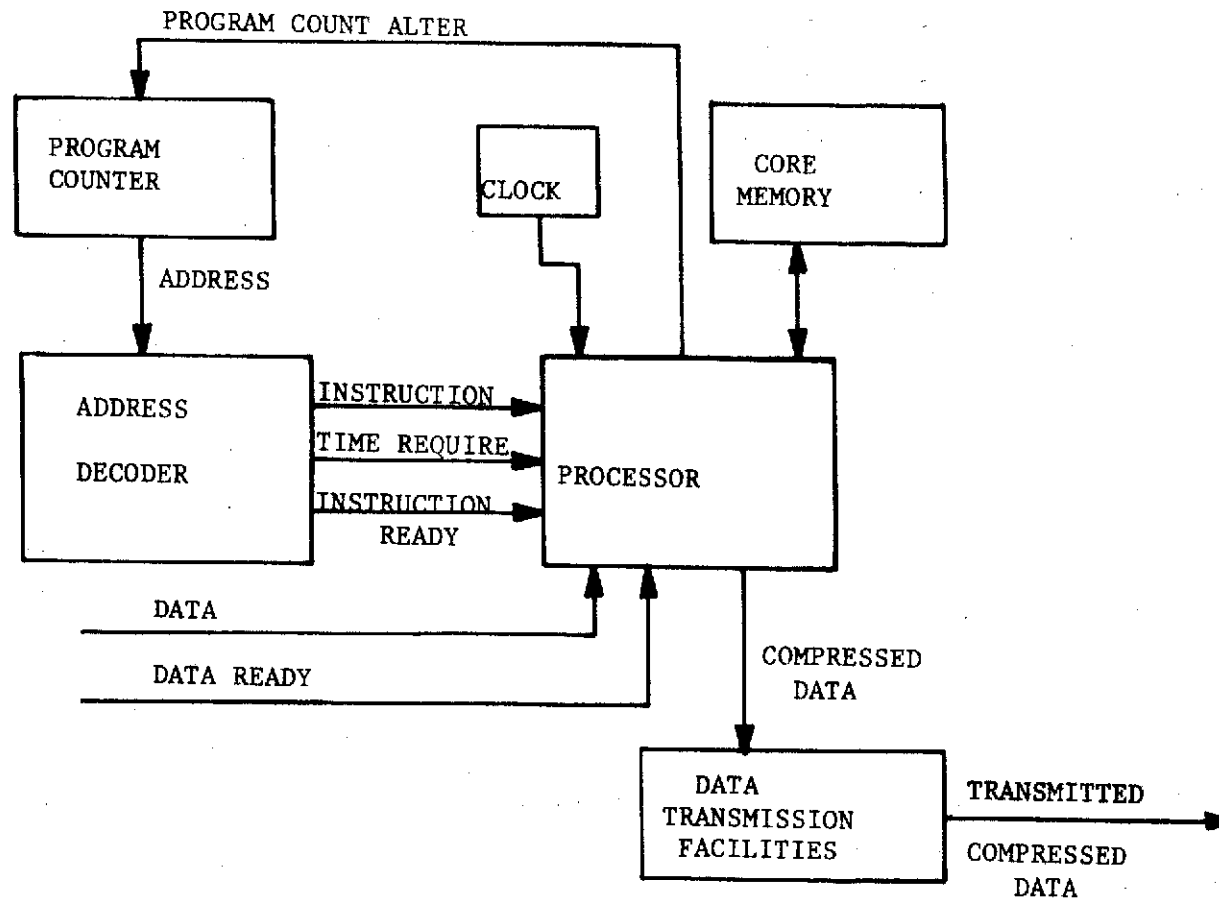


Figure 4-2. Block diagram of a special-purpose, completion countdown data compression computer.

tion. This time requirement is entered into a counter which decrements each time a clock pulse is received. When the counter (sometimes called the "completion counter") reaches zero, the processor has had sufficient time to complete execution and will provide an "execution complete" pulse. A block diagram of a completion count-down system is shown in figure 4-2. This system operates in a manner described as follows.

Initial assumptions: 1) The clock is a free running timing device such as an astable multivibrator, a crystal controlled oscillator, etc.
2) The system is ready to begin analysis of the next data sample (the program counter is set at the location of the last instruction executed).

Typical operation: 1) When a "data ready" pulse is received, the processor sets the program counter to the location of the first instruction in the data compression sequence.
2) Upon arrival of the "instruction ready" pulse the processor begins executing the instruction.
3) Almost immediately upon completion of the instruction (within 1 nsec or so), the processor increments the program counter and awaits the next "instruction ready" pulse.
4) When a transfer or jump is required the processor sets the program counter to the proper location to obtain the next instruction, and awaits the corresponding "instruction ready" pulse.
5) Following the completion of the instruction set, the processor enters a "wait" mode until the next "data ready" pulse is received.

Steps 1) through 5) are then repeated.

Since the system of Figure 4-2 makes quite efficient use of its time, its speed at this point is dominated by such factors as processor speed, the data compression algorithm employed, and the cycle time of the core memory. The core memory, of course, is necessary since it provides storage for pre-flight programming as well as providing the buffer storage for the retained data samples.

C. Hardware Requirements

The hardware required to implement data compression will, of course, vary with several factors. One of the main influencing factors is the data compression algorithm which is to be realized. Naturally, the more complicated the algorithm, the more the hardware that will be required for implementation.

In Chapter II, there were four data compression algorithms mentioned as possible candidates for the system to be realized. Table 4-1 shows a tabulation of hardware requirements and typical execution or cycle times for the algorithms of interest [6].

Algorithm	TTL IC Package Count	Cycle Time	Number of 8-bit Reference Words per Channel
ZOP	50	5 μ sec	2
ZOI	87	6.5 μ sec	3
FOP	62	6.0 μ sec	3
FOI	202	20.0 μ sec	8

Table 4-1. Typical hardware requirements and cycle times for different algorithms.

The values in Table 4-1 were calculated on the basis of the following assumptions [6].

- 1) The data word entering the compressor is eight bits long.
- 2) The core memory has a cycle time of no more than 2 μ sec.
- 3) For an eight-bit word an ADD time of 250 nsec (with present technology, this estimate is quite conservative).
- 4) The system clock operates at 2 MHz.
- 5) DIVIDE operations are assumed to require N clock pulses, where N is the maximum number of bits in the word.
- 6) In most cases, where numbers are to be divided or multiplied by an integral power of 2, it is done on a wired basis in such a way that no extra clock pulses are required.
- 7) Reference values needed by an algorithm on a per channel basis are assumed to be eight bits long and are stored in thirty-two bit blocks.
- 8) Apertures (corridor values) are stored rather than wired, so that adaptive aperture (variable corridor) techniques may be applied.

It is quite obvious from the table that the ZOI or ZVA algorithm suffers a disadvantage in both cycle time and the number of IC packages required for implementation. As mentioned in Chapter II, however, these disadvantages are thought to be outweighed by the gains in performance realized by the ZOI algorithm. There are two major reasons for this line of thought. First, the additional circuitry required to implement the ZOI algorithm should not add appreciably to

the size of the system. Furthermore, the use of LSI circuitry could drastically reduce the amount of external wiring as well as the number of IC packages required, further reducing the overall size of the system. Second, the maximum value of W (see definition 3-1 in Chapter III) for the system is the inverse of the cycle time, or, in the case of the ZOI algorithm,

$$W = \frac{1}{6.5 \mu\text{sec/sample}} \approx 153 \text{ K samples/sec.}$$

However, it has been determined that the maximum required sampling rate for any one channel should be 120 samples/sec [3]. Assuming, for the purposes of a worst-case analysis, that all sixty-four channels must be operated at the maximum sampling rate, the minimum allowable value for W is given by

$$W = (\text{sample rate}) \times (\text{number of channels}),$$

$$\text{or } W = (120 \text{ samples/sec-channel})(64 \text{ channels}),$$

which simply means that $W = 7680 \text{ samples/sec.}$ By no means will the rate put a strain on a system capable of operating at 153 K samples/sec.

D. Cost

It would certainly be logical to assume that as the complexity of a special-purpose machine (or any machine for that matter) increased, so would the cost. This being the case, one must consider the design requirements very carefully before choosing any of the current special purpose data compressors. Furthermore, it must be decided as to whether or

not it would be beneficial to design a new machine (the cost of design is also a direct expense) if it has been determined that significant production and weight costs may be saved over existing machines. Suppose, for example, that there are several multi-algorithm machines available, but only a single algorithm machine is required. If the least expensive machine available cost \$60,000, and it would cost \$60,000 to design, debug, and produce a single algorithm machine, then it might be wise to switch to the new design. There are three main reasons behind this choice. First, being much simpler, the machine would weigh less and require a smaller amount of space. Second, also due to simplicity, power requirements would probably decrease and reliability would likely increase. Third, the design costs tend to make the purchase cost of the first few production machines quite high. As more machines are produced, however, the cost per machine begins to decrease quite sharply and it will eventually level off at the then current per machine production cost (i.e. all cost incurred, at this time, is production cost). To be sure, there will be little decrease in cost per unit if the production is limited to a very few (four or five) units. However a new design may be desirable simply from the standpoints of weight, size, power consumption, and reliability, if costs are comparable. Figure 4-3 shows an approximate cost spread for special-purpose data compression machinery. [5]. This curve is based on machines which may possess multi-algorithm or single algorithm capabilities, but only simple (ZOP, ZOI, FOP, FOI) algorithms are considered. Naturally, there would be an increase across the board for the more complex algorithms.

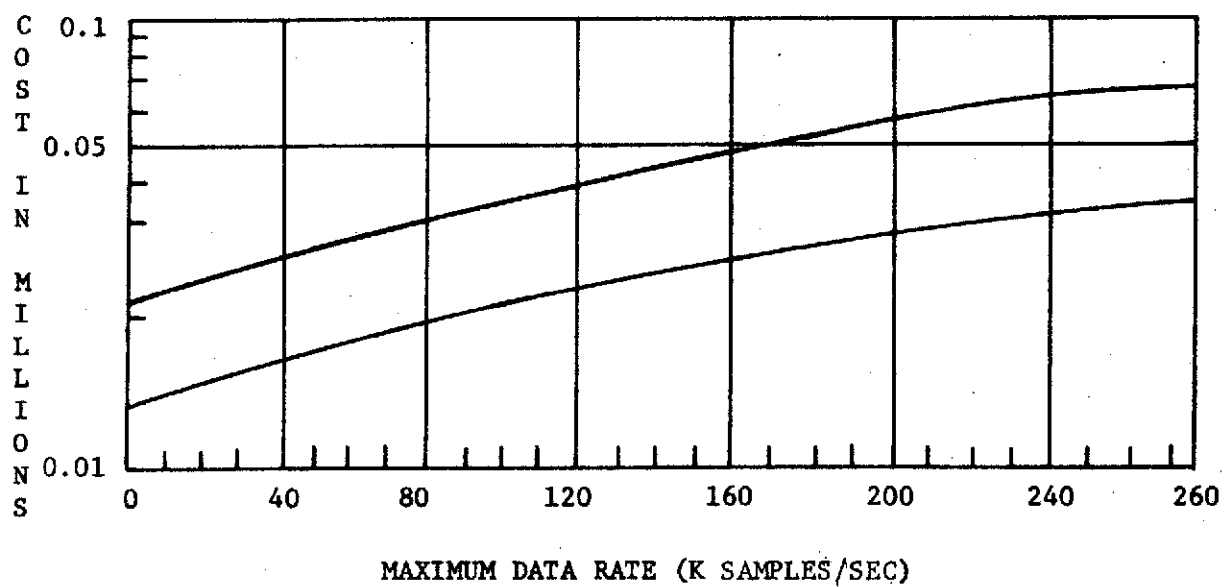


Figure 4-3. Approximate cost spread vs. sample rate for special-purpose hardware.

E. The TDC-1 Special-Purpose Data Compression Computer [10]

Since the increase of interest in data compression which came about in the mid-sixties, there have been a number of experimental and operational data compressors designed and built. One in particular quite closely approximates the specifications given in Chapter III and it will, therefore, serve as an example of the type of special purpose data compression computers which are currently available. This machine is the Lockheed TDC-1 data compressor. It might be mentioned, before beginning the discussion, that some of the terminology found in the Lockheed operations manual has been altered slightly so as to fit the definitions introduced in section B of Chapter III. For example, the term "master frame" is used in the manual, but it corresponds to the term "main frame" introduced in Chapter III. Hence, "main frame" will be used throughout the discussion.

Input data - The TDC-1 processes input data having rates and format as described below.

- 1) The commutation rate is 7200 or 14,400 samples per second, corresponding to a system clock rate of 144 KHz or 288 KHz, respectively.
- 2) Channel sampling rates are pre-programmable at 4, 12, 40, and 120 samples per second.
- 3) Main frame - A main frame consists of 30 sub-frames.
- 4) Sub-frames - A subframe consists of either 60 or 120 consecutive subframe time slots, corresponding to system clock rates of 144

KHz and 288 KHz, respectively.

- 5) Sub-frame time slots - Input data will be available for examination by the data compressor at prescribed time slots relative to the leading edge of the main frame sync pulse. Each time slot has a duration of about 20 system clock pulses.
- 6) Sub-frame time slot groups- The time slots within a sub-frame may be considered to be grouped into time slot groups in a manner such that the sampling rate may vary from channel to channel. This is accomplished by the following time slot grouping.

<u>Group</u>	<u>Time slot position</u>
I	$2N + 1$
II	$2N + 2$
III	$4N + 1$
IV	$4N + 2$
V	$4N + 3$
VI	$4N + 4$
VII	Last three time slots of sub-frame

Here, N is an integer and ranges upward from zero to the number which corresponds to the maximum allowable number of time slots, within a subframe, which may contain active data samples. The term "time slot position" simply refers to the position of the time slot with respect to the beginning of the sub-frame. Time slot groups I, II, and VII are used when the system clock runs at 144 KHz, while time slot groups III, IV, V, VI, and VII are

in force when a system clock rate of 288 KHz is used. It should be noted, however, that group VII does not contain any information to be processed.

- 5) Combinations of channel sampling rates - The data within each time slot may be obtained with any one of four sampling rates. However, data associated with the same time slot group must all be sampled at 12 and/or 120 SPS, or all be sampled at 4 and/or 40 SPS.

Output data - The TDC-1 data compressor provides outputs described as follows.

- 1) A compressed PCM signal is provided, and is in NRZ-Space format.
- 2) The PCM data word contains 10 magnitude bits, plus a time slot identification of 6 or 7 bits, as required.
- 3) The PCM synchronization word is either 32 bits long (system clock frequency of 144 KHz), or 34 bits (system clock frequency of 288 KHz), and is output once every 66 output word times.
- 4) The frame sync word is output every multiplexer sub-frame. This word serves to identify the specific multiplexer sub-frame within a main frame sequence, and to identify the main frame within a sequence of 16 main frames.
- 5) A buffer status word is output every 16 multiplexer main frames. This word indicates the level of buffer occupancy at the time it is entered into the buffer.
- 6) The PCM output bit rate is preflight programmable, and is defined by the following equation

$$B = \frac{Fc}{2n}, \text{ where}$$

B is the bit rate, Fc is the system clock frequency, and n is the programmable integer parameter which may range from 3 to 16.

Data compression algorithm - The zero-order, variable corridor prediction algorithm is used to determine redundancy of data samples, with the prediction tolerance limits being assigned, through the command input, independently for each channel. There are six possible tolerances which might be chosen. These tolerances, in terms of percent of the 10 bit data word are shown in Table 4-2, together with the significant bits for each tolerance.

Programmable control logic - Programmable control logic is provided so that data acceptance may be controlled by a set of input logic signals.

- 1) Control logic inputs - The following control signals are pro-

Tolerance	Significant bits
+0.1%	1 - 10
+0.78%	4 - 10
+1.56%	5 - 10
+3.13%	6 - 10
+6.25%	7 - 10
+12.5%	8 - 10

Bit 1 = LSB , Bit 10 = MSB

Table 4-2. Available tolerance ranges and correspond significant data bits for TDC-1 data compressor.

vided, within the TDC-1, as inputs to the control logic.

- a) Command (from vehicle command system) to force acceptance.
 - b) Two signals which indicate two specific combinations of prediction tolerance limits ($\bar{T}_0 \cdot \bar{T}_1 \cdot \bar{T}_2$, and $T_0 \cdot T_1 \cdot T_2$).
 - c) A stored priority tag
 - d) A stored redundant sample tag
 - e) Inputs for the six ranges of buffer occupancy status.
 - f) Four inputs, each corresponding to a specific sampling rate.
 - g) Four inputs, each corresponding to the time slot group to which the data sample being examined belongs.
- 2) Control logic output functions - The application of the inputs previously described allows the control logic to realize the following functions.
- a) Normal algorithm operation with nominal tolerance.
 - b) Normal algorithm operation with twice the nominal tolerance.
 - c) Normal algorithm operation with four times the nominal tolerance.
 - d) Force acceptance of sample.
 - e) Force rejection of sample.
- 3) Fixed control logic functions - The following conditions are those which always produce a specific control logic output function.

- a) When buffer occupancy reaches the minimum allowable level, all data samples are accepted.
- b) When buffer occupancy reaches the maximum allowable level, all data samples will be discarded, except for those samples processed concurrently with a commanded "force accept" input.
- c) One (1) data sample will be accepted from any channel addressed with a "force accept" command.

Commanded functions - There are certain commands available which are used to perform various functions of control. A command is executed when its corresponding input code is received from the vehicle command system. Commands are available to perform the following functions.

- 1) Change stored prediction tolerance values assigned to the addressed channel.
- 2) Change stored priority tag assigned to the addressed channel.
- 3) Force acceptance of one data sample from the addressed channel.

Confidence sampling provisions - The TDC-1 provides for acceptance of redundant data samples to prevent buffer underflow, as well as providing confirming samples for semi-static channels.

Buffer memory - A buffer memory is provided of length 1024 words. Also provided is a means to constantly monitor the level of buffer occupancy which will detect 6 occupancy levels within the buffer. The lowest level is set at 4 words, with the remaining five levels being programmable in increments of 16 words or less.

As can be observed from the foregoing discussion, the TDC-1, while closely approximating the desired design features, still has some shortcomings. First, and probably foremost, is the fact that the TDC-1 uses the ZOP algorithm, and it is difficult to say exactly how much trouble would be involved in converting to the ZOI algorithm. Second, there is a deficiency in the number of available sampling rates, since the TDC-1 has only four rates available. Once again, this could probably be altered to meet the sampling rate requirements as specified in Chapter III, but the lack of versatility makes alterations to the system operating procedure quite difficult. If it is deemed that a special purpose machine be used, it may be easier and less expensive to seek a new design which exactly matches the specifications.

The difficulty in reprogramming the special purpose computer logically leads to the question of whether or not a general purpose computer might be better suited to this application. It is this question that is the subject of the following chapter.

V. Data Compression Implementation: The General-Purpose Computer

When the concept of data compression was first introduced with its varied compression algorithms, the general-purpose computer, because of its versatility and ease of programming, seemed to be a natural choice for implementation of the idea. At this time, however, general-purpose computers were either too expensive or too slow for most applications, so special-purpose data compression computers were employed. Subsequently, general-purpose computer technology began taking great strides forward and, as a result, the general-purpose computer became faster and less expensive, hence becoming practical for data compression implementation. Recently, however, the quantity of data to be handled has increased by such a great amount that the price of general-purpose computers with sufficient capacity has once again become prohibitive. This has resulted in the present trend back toward the use of special-purpose computers for data compression implementation.

Since there are several factors which must be considered, such as speed, cost, versatility, reliability, etc., a fairly accurate analysis of the capabilities of available systems must be made before a decision to use either a general-purpose computer or a special-purpose computer is reached. In Chapter IV special-purpose computers were discussed, with emphasis on a particular machine. Similarly, this chapter is devoted to the discussion of the features of general-purpose computers, with a section allocated to a specific machine.

As previously mentioned, the general-purpose computer is quite versatile, having the ability to be programmed for almost any contingency from simple accounting problems to the most complex mathematical calculations. This versatility stems from the fact that the instruction sequence or "program" used by the computer is stored in a medium which allows the programmer to easily add instructions, remove instructions, or otherwise edit the program. Before proceeding, and in order to better understand the limitations of general-purpose computers, a brief discussion of main memory systems is in order.

B. Main Memory Media

In any literature concerning main memory systems, certain terms are frequently encountered in the memory system specifications. To aid in comprehending these terms, the following definitions are made.

Definition 5-1. Memory Cycle - all operations which are required within the memory unit when a "read" command or a "write" command is received.

Definition 5-2. Cycle Time - the time required to complete one memory cycle.

Definition 5-3. Read Phase - the phase of the memory cycle during which information is read from a specified memory location.

Definition 5-4. Write Phase - the phase of the memory cycle during which information is written into a specified memory location.

Definition 5-5. Memory Address Buffer - a register which contains the address of the location in the memory where the read or write phase takes place.

Definition 5-6. Memory Data Buffer - a register which contains the information or data which has been read from or written into a specified memory location.

Definition 5-7. Destructive Readout - a read process which destroys or erases the data contained in a specified memory location as the data is read into the memory data buffer.

Definition 5-8. Non-destructive Readout - a read process which does not alter the data contained in a specified memory location as the data is read into the memory data buffer.

Definition 5-9. Access Time - the time which elapses between the time a "read" command is received and the time the data is available in the memory buffer, or the time which elapses between the time a "write" command is received and the time the data in the memory data buffer has been written into memory (in most literature there is no distinguishment between "read access time" and "write access time"; the term "access time" is used for both).

There are several types of memory media available, with varying costs, access times, and cycle times. Although the purpose of this paper is not to analyze the performance of various memory systems available, a few descriptive remarks may aid in a better understanding of subsequent topics.

Almost all of the memory systems available today may be grouped into three broad categories based on the media employed. These categories are, 1) dynamic magnetic systems (magnetic tape, discs, drums), 2) static

magnetic systems (core), and 3), electronic systems (systems using bipolar or MOS devices). Dynamic magnetic systems are characterized by non-destructive readout, high storage capacity, and rather slow access times (on the order of milliseconds). These systems are usually restricted in use to backup memory systems due to lack of speed. Static magnetic systems are characterized by destructive readout, relatively limited storage density, and fairly fast cycle times (0.8 μ sec - 3 μ sec typ.). The magnetic core is the best example of this technology, having proven its ruggedness and reliability over the past decade. Mass production techniques have, over the years, lowered the cost of core memories, hence economy is an additional dividend. Electronic systems are characterized by non-destructive readout (in general), fairly high storage densities, and fairly fast (MOS devices) to extremely fast (bipolar devices) access times (50 nsec - 2 μ sec typ). With the present state of the art, electronic memories are still, bit for bit, a good deal more expensive than core memories. An additional disadvantage of electronic systems is the fact that power must be maintained in order to assure the integrity of the data contained in the memory. Any momentary power failure usually results in complete loss of data in electronic memory systems. Both core memory systems and electronic memory systems are quite suitable as main frame memories in general-purpose computers, but core memories are, by far and away, the most popular of the two.

It might be mentioned, in passing, that there are two more types of static magnetic memory systems which promise increased speed and/or

increased storage density. These are 1) magnetic bubble, and 2) magnetic domain tip. Both of these concepts are still experimental and it is likely to be several years before they will be proven reliable or will bear a reasonable price tag. The same facts are true of the present experimental holographic memory systems.

Since this chapter will have a section devoted to a particular general-purpose computer, it would be reasonable to restrict the remaining discussion in this section to the type of main memory system employed by this computer, namely core memory.

The biggest disadvantage of core memory is the characteristic of destructive readout. In order to guard against loss of data, most core memory systems are set up, using some sort of micro-programming technique, so that both the "read" command and the "write" command will cause the system to complete one full memory cycle. To illustrate, consider the following steps which are required to perform a read function in a core memory system without loss of data.

- 1) Place address of location to be read from in memory address buffer.
- 2) Apply a pulse to the "read" command input.
- * 3) Perform read phase, placing information into memory data buffer (when this phase is completed, the data in the specified location has been erased or destroyed.)
- * 4) Perform write phase, storing the data contained in the memory data buffer back into the specified memory location (this phase

effectively "restores" the data which was originally stored in the specified location.)

- * 5) Send "cycle complete" pulse to central processing unit.

Those steps marked by an asterisk (*) are steps performed within the memory system and are usually under micro-program control. For most core memory systems, the address must be present in the memory address buffer before the read or write command is received by the system, and must remain unaltered until a "cycle complete" pulse is output by the memory system.

A write function is performed in much the same manner, except that during the read phase, the information contained in the specified location is not gated into the memory data buffer. Rather, it is discarded, which renders the read phase as nothing more than a "clear" operation. In some core memory systems the information to be written into memory must be present in the memory data buffer before the write command is received by the memory system, while in others, the information may be placed in the memory data buffer any time before the write phase begins.

From the discussion in this section, two very important facts, regarding core memory systems, may be extracted. First, both the "read" and "write" commands cause the core memory system to complete one full memory cycle. Second, once a "read" or "write" command pulse is received by the system, it must complete one memory cycle before it can process another command. To be sure, when a read command is executed,

the data is available in the memory data buffer after the access time has elapsed, but the data in the buffer may not be altered because it has not been restored by the write phase of the memory cycle. These facts should be kept in mind while reading the next section, which deals with the speed of general purpose computers.

C. Speed

Although the various main memory systems previously discussed lend extreme versatility to the general-purpose computer, they do, in general, cause a severe decrease in the speed with which the computer operates. The main reason for this is the fact that the computer must "read" or "fetch" each instruction from memory, one by one, as the program is executed. Simply stated, this means that for each instruction to be executed in the program, at least one memory cycle must be executed.

There is one class of instructions which require only one memory cycle (that is to fetch the instruction itself). These are usually called "generic" instructions and they describe an operation which is to be performed in a register, say, such as complementing the register, shifting right or left, etc. Other classes of instruction, such as input-output instructions and memory reference instructions, in general, require more than one memory cycle for execution.

In addition to the time required for the memory cycle, there is a given amount of time required for the processor (arithmetic units, shift registers, program counter registers, etc.) to perform the tasks which

are required of them for each instruction to be executed. Since most of these processor components are bipolar electronic devices or bipolar integrated circuits, however, this processor cycle time is fairly small when compared to the cycle time of even the fastest core memory (there are certain instructions, nonetheless, which will cause the processor to use an amount of time significantly greater than the memory cycle time, such as multiply or divide instructions).

If a programmer was well familiar with the machine for which he was writing a program, that is, if he knew the number of memory cycles required for each of the various instructions, as well as the processor cycle time for each instruction, then it would be a rather simple matter to sum the times required for each instruction in the program to obtain the time needed to make one iteration of the program. The number of iterations per second, then, is the reciprocal of the time required for one iteration. However, the process of familiarizing oneself with the characteristics peculiar to a given machine can be quite bothersome and time consuming, since processor cycle times may vary quite a bit from instruction to instruction. Also, as previously mentioned, the number of memory cycles may vary between instructions, but this variance is not so great, generally ranging from 1 to 3 memory cycles per instruction. In most cases, the programmer need not have to learn all these details before being able to calculate the iteration time of program, because most computer manufacturers supply, with their software documentation, a table of execution times for each instruction in the set used by the

particular machine to which the software applies. As will be discussed in subsequent sections, determining the iteration time for a given program may not be as simple as implied in this section, due to the possibility of branches within a program, but a method will be shown as to how to determine the safest minimum iteration time for a given program.

D. Programming Techniques

If three programmers were to each write a computer program so that all three programs performed exactly the same function or functions, then the odds are quite high that 1) no two of the programs would require the same amount of storage and 2) no two of the programs would have the same iteration time. What this means is that a programmer must decide whether his program is to be designed to occupy the least amount of memory in the computer or to require the least amount of iteration time or to be a compromise between memory requirements and execution time. This dilemma most often exists in programs where there exists, along with the instruction set, a data set which is to be processed, in some way, by the instruction set. It is this type of program which is the subject of the following discussion.

Consider a program which consists of an instruction set plus 1) N data entries which each require only one bit of storage, 2) K data entries which each require seven bits of storage, and 3) M data entries which require sixteen bits of storage each. Further suppose that the computer on which this program is to be run is a sixteen-bit machine. The programmer has two

basic options open to him at this point. He could allow each data entry to occupy one word (16 bits) of memory, in which case the data storage requirements would be

$$K + N + M \quad \text{words} \quad .$$

Data stored in this manner is quite often referred to as unpacked data. On the other hand, the programmer might elect to use packed data. For example, if there are N one-bit data entries, then one word could provide storage for sixteen of these data entries. Another method of "packing" the data would be to let one word provide storage for two of the seven-bit entries and two of the one bit entries. The object of packed data is to make the most efficient use of the space where the data is to be stored. It is quite obvious that where there is a large number of data entries, each requiring fewer bits than the standard word in the computer, significant reductions in storage requirements may be realized by using packed data. Of course, as the number of bits per data entry increases, the significance of the storage requirement reduction, using packed data, decreases.

To further illustrate the advantage to using packed data, consider again the previous example where $N = 28$, $K = 28$, and $M = 5$. If unpacked data is used, then the data storage requirement is

$$28 + 28 + 5 \quad \text{or} \quad 61 \text{ words}$$

It is quite apparent that no reduction can be realized where the sixteen-bit data entries are concerned. However, it is noted that the total number of bits required for the storage of the one-bit data entries is

28 bits, while 7×28 or 196 bits are required for the seven-bit entries. There are many ways in which the data might be packed, but only two will be illustrated. The first way would be to place 14 one bit entries in each of 2 words and then to place 2 seven bit entries into each of 14 words. In this way, the total data storage requirement is

$$2 + 14 + 5, \quad \text{or} \quad 21 \text{ words.}$$

The second way would be to place 2 seven-bit data entries and 2 one-bit data entries into each of 14 words. This would bring the total data storage requirement to

$$14 + 5, \quad \text{or} \quad 19 \text{ words.}$$

It is rather easy to see that either of the afore mentioned ways to pack the data would cause a significant decrease in data storage requirements.

Using packed data does, nonetheless, introduce some problems, all of which stem from the fact that the data stored in this form requires a process by which the desired information is extracted from the packed data word. This process is sometimes referred to as "unpacking" or "decoding". As with any other process which a computer is required to perform, an instruction set must be provided to perform this decoding of the packed word. This instruction set is obviously going to require storage, but more important, it is going to demand a certain amount of execution time for each decoding operation required. Naturally, when the data has been packed in some symmetrical manner so as to reduce the

number of instructions required for decoding, the execution time for each decoding operation will be reduced. However, when many decoding operations are required, there will likely be a significant increase, in the amount of time required to realize the same functions, over the time needed when unpacked data is employed.

If memory space is a critical factor and there is a fairly large amount of data requiring fewer bits than the standard word in the particular computer, it would probably be best to use packed data, at the expense of processing time. By contrast, if speed is the critical factor and there is memory space available, then unpacked data should be used. It may be decided that a compromise is necessary, but the final decision would depend, of course, on the nature of the function which the computer is to perform.

Many of the newer general-purpose computers provide additional hardware features, such as overlapped core storage, instruction look-ahead, parallel execution, and high-speed scratch-pad memories. Each instruction is, in effect, fetched or read from memory whenever possible during the execution of the previous instruction. In some cases, multiple index registers and accumulators are available to allow parallel execution whenever possible. All of these hardware features make the programmer's task much easier as well as offering potentially great computational power. However, it is next to impossible to realize this potential, in most cases, due to the existence of branching instructions, which render instruction look-ahead and parallel execution useless. This

is because the previous instruction must be fully executed before it is known where the next instruction to be executed is located [8]. Since most of the smaller flight computers do not possess such features, however, there will be no more discussion on the subject, except to mention that when considering a ground based data compression installation or even a similar installation in an orbiting space station, these larger general-purpose computers might be quite desirable due to increased capacity.

E. The Critical Path

The concept of the critical path has been around for quite few years, having found application in the planning of a wide range of projects (construction projects, in particular) which demand certain resources, such as time and money, which may, at one point or another, become a critical factor. The idea of the critical path may be stated as follows.

For any given process composed of multiple paths, each involving a different set of steps for completion, there will exist at least one path, which will require the greatest expenditure of resources for completion.

Although it is possible for more than one critical path to exist, this is very rarely the case. Also, there is no requirement as to whether or not all paths must be completed each time the process is completed, i.e., the process may be performed via one path under one set of conditions, while under other circumstances a different path may be taken. The critical resource is also a significant factor in determining the critical path. For example a path which is critical by

time may not be critical with respect to another resource, like equipment. Since one of the main concerns in subsequent sections of this chapter will be the speed with which a data compression program can be executed, time will logically serve as the critical resource.

Because a flow chart might greatly aid in determining the critical path of a process, the next section will deal with the flow chart of the data compression procedure outlined in Section I of Chapter III.

F. System Operation Flow Chart

Before proceeding with a presentation of a typical system operation flow chart, it would probably be helpful to briefly define the variable names which are used therein. These definitions are presented below.

TEMP - a location in memory where the data sample is temporarily stored during redundancy analysis.

OSC - the output synchronization count; this variable is incremented each time a data ready pulse is received and causes a word to be output from the buffer when it becomes equal to R, at which time OSC is set to zero.

BFAD1 - the address in the buffer where non-redundant data samples are stored.

BFAD2 - the address in the buffer from which data is extracted for transmission.

R - the number of data ready pulses which must be received before a word is output from the buffer.

- TSN - time slot number; indicates the time slot or channel address which is currently being tested. $0 \leq \text{TSN} \leq 63$.
- SFN - sub-frame number; indicates subframe currently being processed. $0 \leq \text{SFN} \leq 31$.
- MFC - main frame count; increments each time the SFN is reset and causes a reading of buffer occupancy to be placed into the buffer (as specified by BOIC) when it reaches a value of 16, at which time it is reset to 1.
- BOS - buffer occupancy status; indicates the number of words in the buffer which contain data for transmission, i.e., the level of buffer occupancy. $0 \leq \text{BOS} \leq 1023$.
- BOIC - buffer occupancy information control; an eleven-bit word which specifies the SFN and the TSN during which the reading of buffer occupancy is to be placed into the buffer for transmission.
- * SC - sample count; indicates the number of samples presented by a particular channel and causes a sample from the channel to be examined when it becomes equal to SRI.
 - * SRI - sample rate index; determine the number of samples which a channel must present before one is except for redundancy analysis.
 - * RDC - rejected data count; increments each time a data sample is determined to be redundant and causes a sample to be retained, regardless of redundancy, when it (RDC) reaches 31, at which time it is reset to zero

ADC - accepted data count; increments each time a sample is determined to be non-redundant and causes the synchronization word (SW) or its complement to be placed into the buffer when it (ADC) becomes equal to 64, at which time it is reset to zero.

SW - synchronization word; this word, or its complement is placed into the buffer after each 64 retained samples.

q1, q2, . . . , q6 - these are the buffer queue control points which are used in determining TM for redundancy analysis.

TN - nominal tolerance; used in determining TM.

TM - modified tolerance; the tolerance which is actually used in redundancy analysis.

* CB - confidence sampling bit; indicates whether or not a particular channel uses confidence sampling.

* PB - priority bit; indicates whether a particular channel is high priority or low priority.

* UCB - upper corridor boundry; indicates upper most corridor boundry resulting from past redundancy analysis.

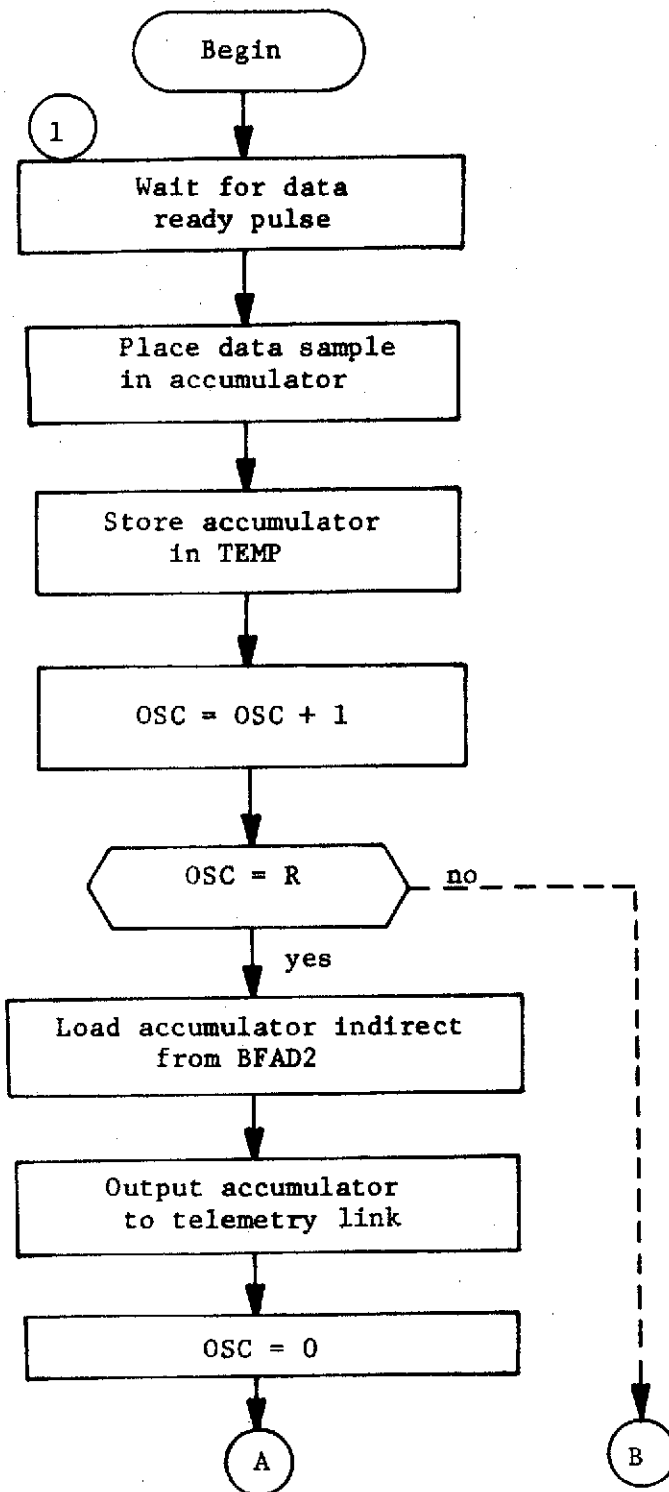
* LCB - lower corridor boundry; indicates lower most corridor boundry resulting from past redundancy analysis.

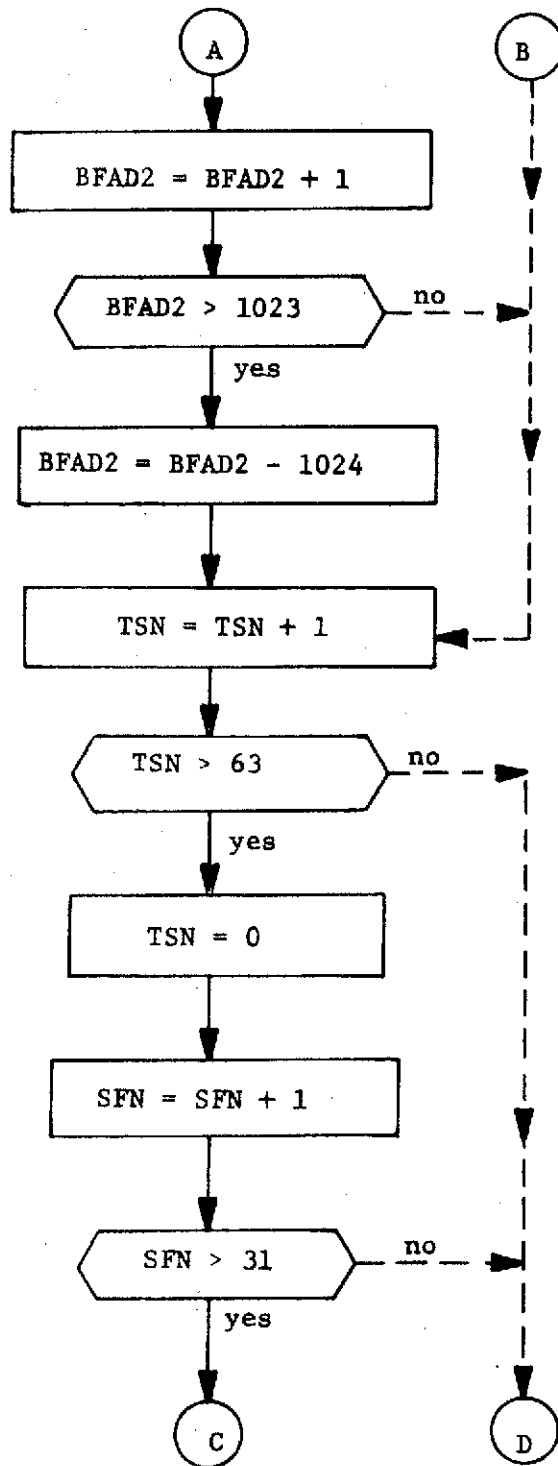
UTB - upper test boundry; the upper most boundry of the tolerance range placed about the data sample during redundancy analysis.

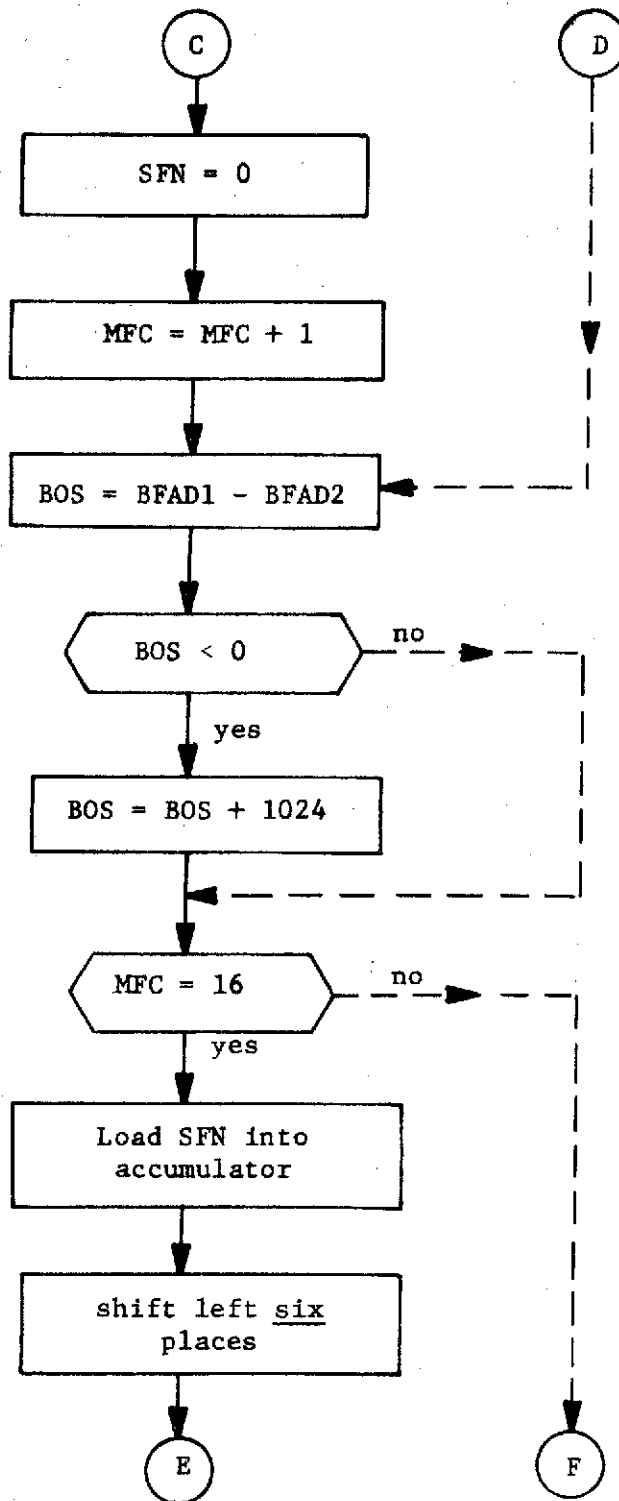
LTB - lower test boundry; the lower most boundry of the tolerance range placed about the data sample during redundancy analysis.

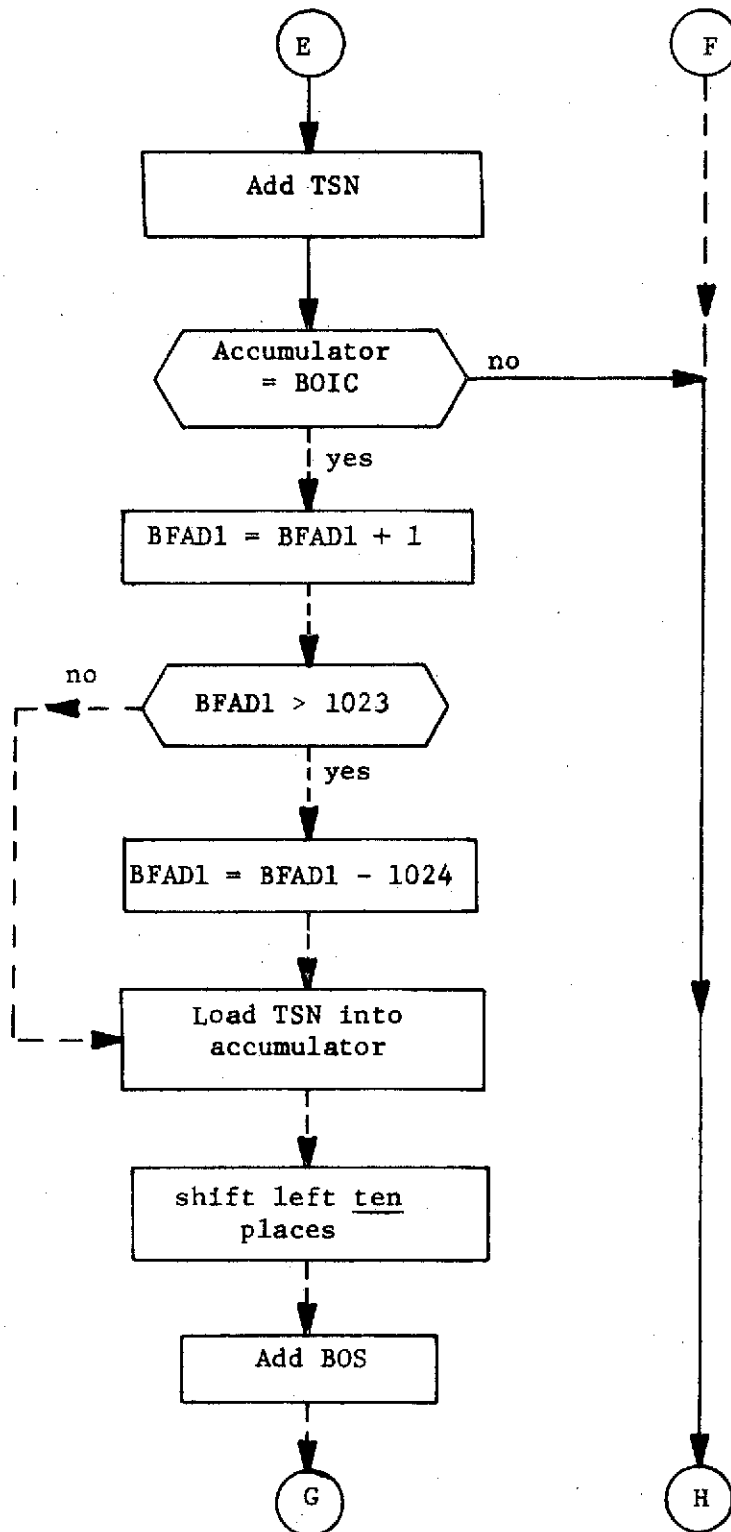
As is quite obvious, some of the terms just discussed have appeared in previous chapters. Those terms marked with an asterisk (*) denote values which are only applicable to one channel, that is, a confidence bit (CB) is stored for each channel, as is a priority bit (PB), a sample count (SC), etc.

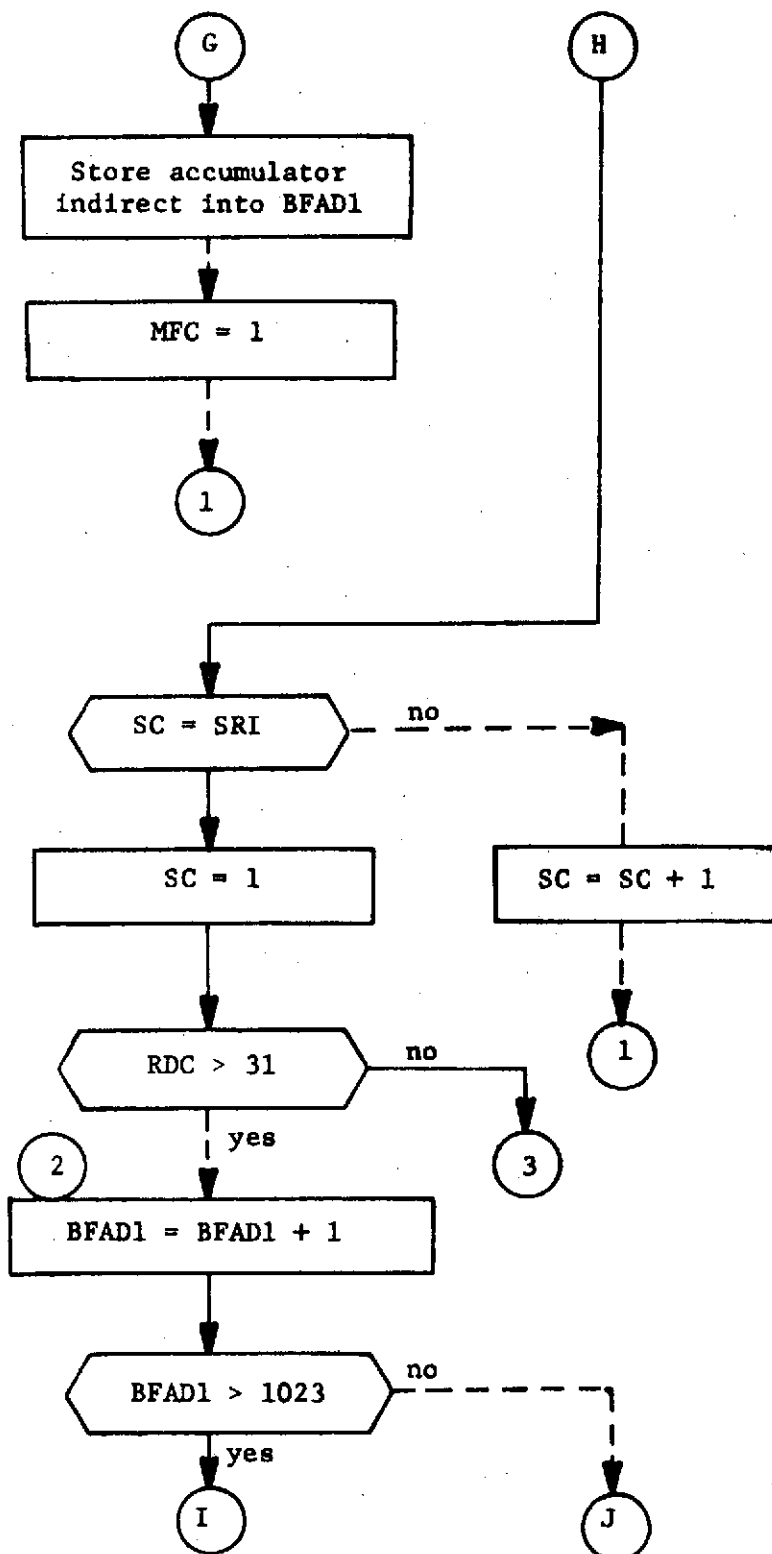
The system operation flowchart assumes 1) that unpacked data format is used and 2) that the buffer storage begins at location 0 decimal and ends at location 1023 decimal in the computer memory. With no further delay, the system operation flowchart may now be presented, beginning on the following page.

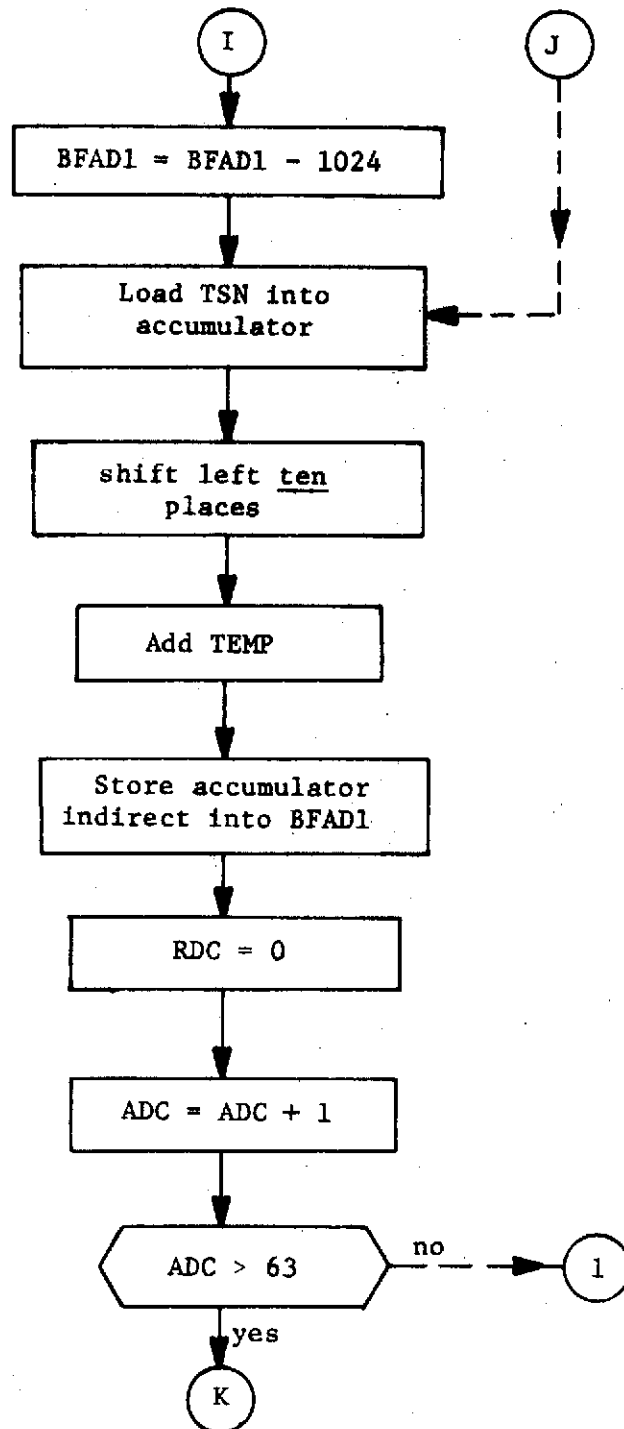


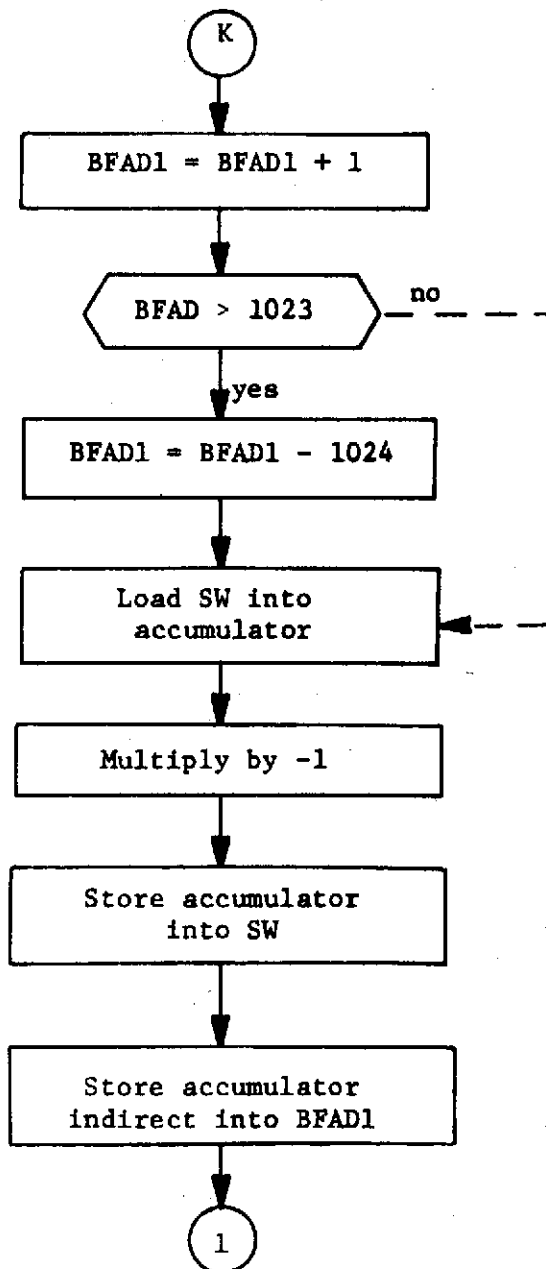


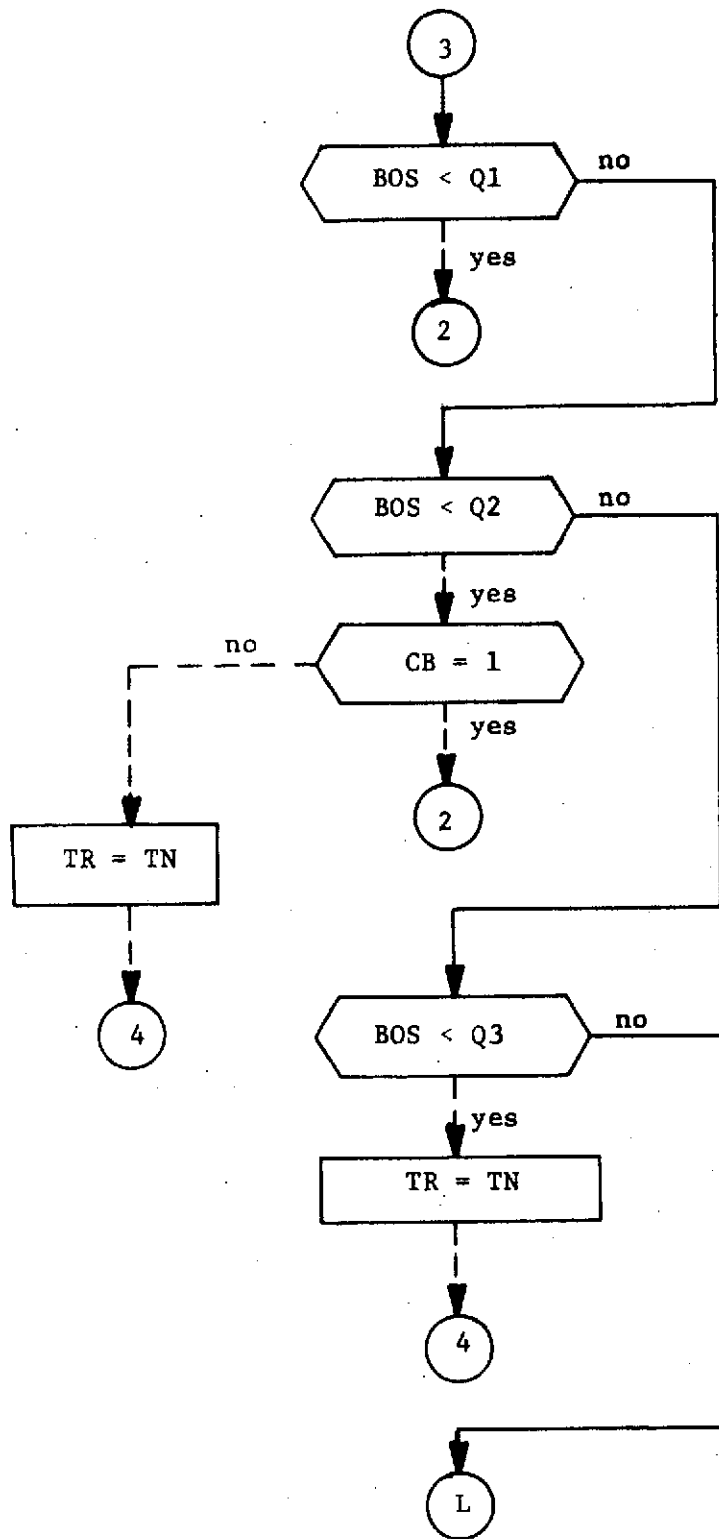


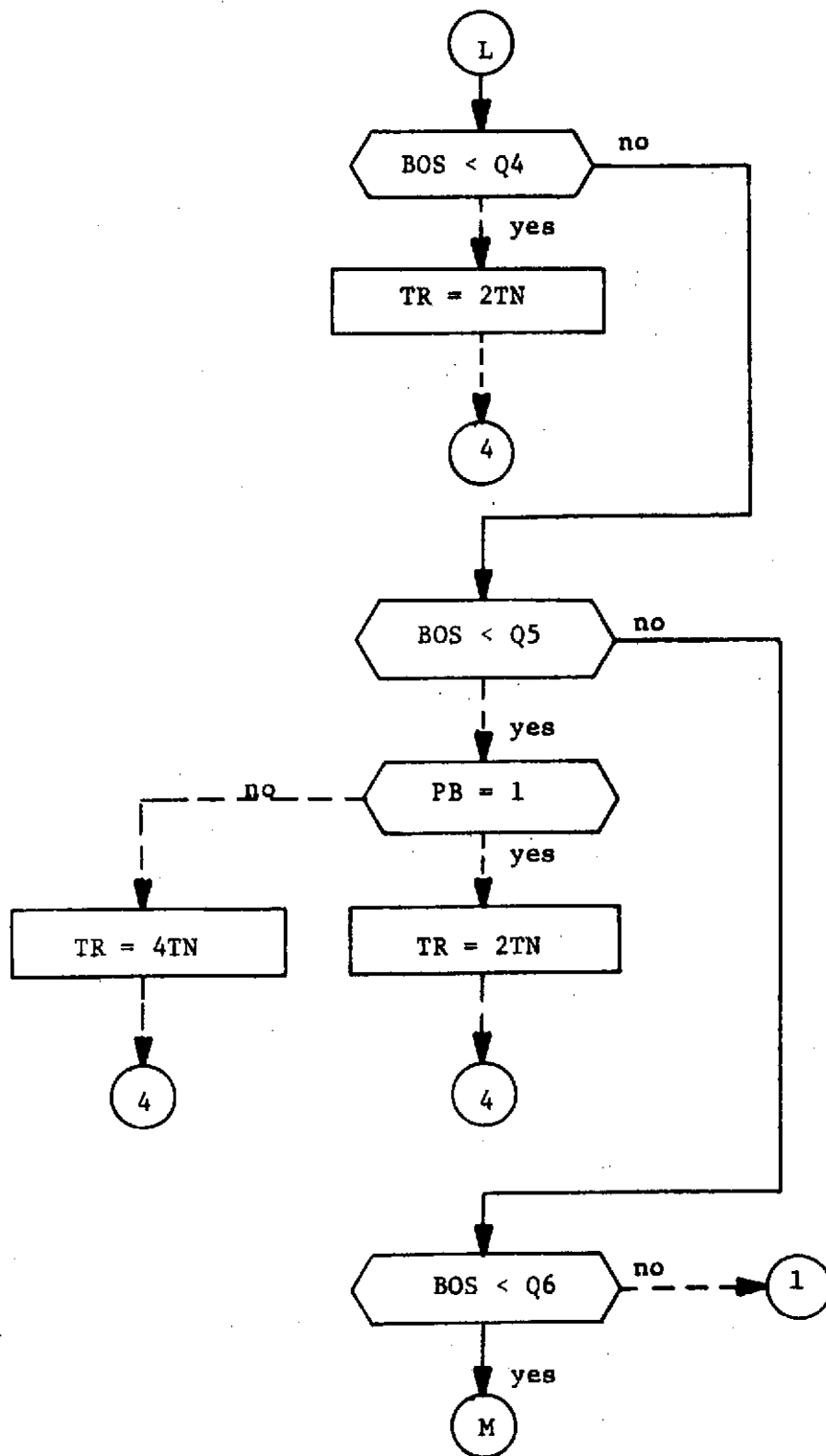


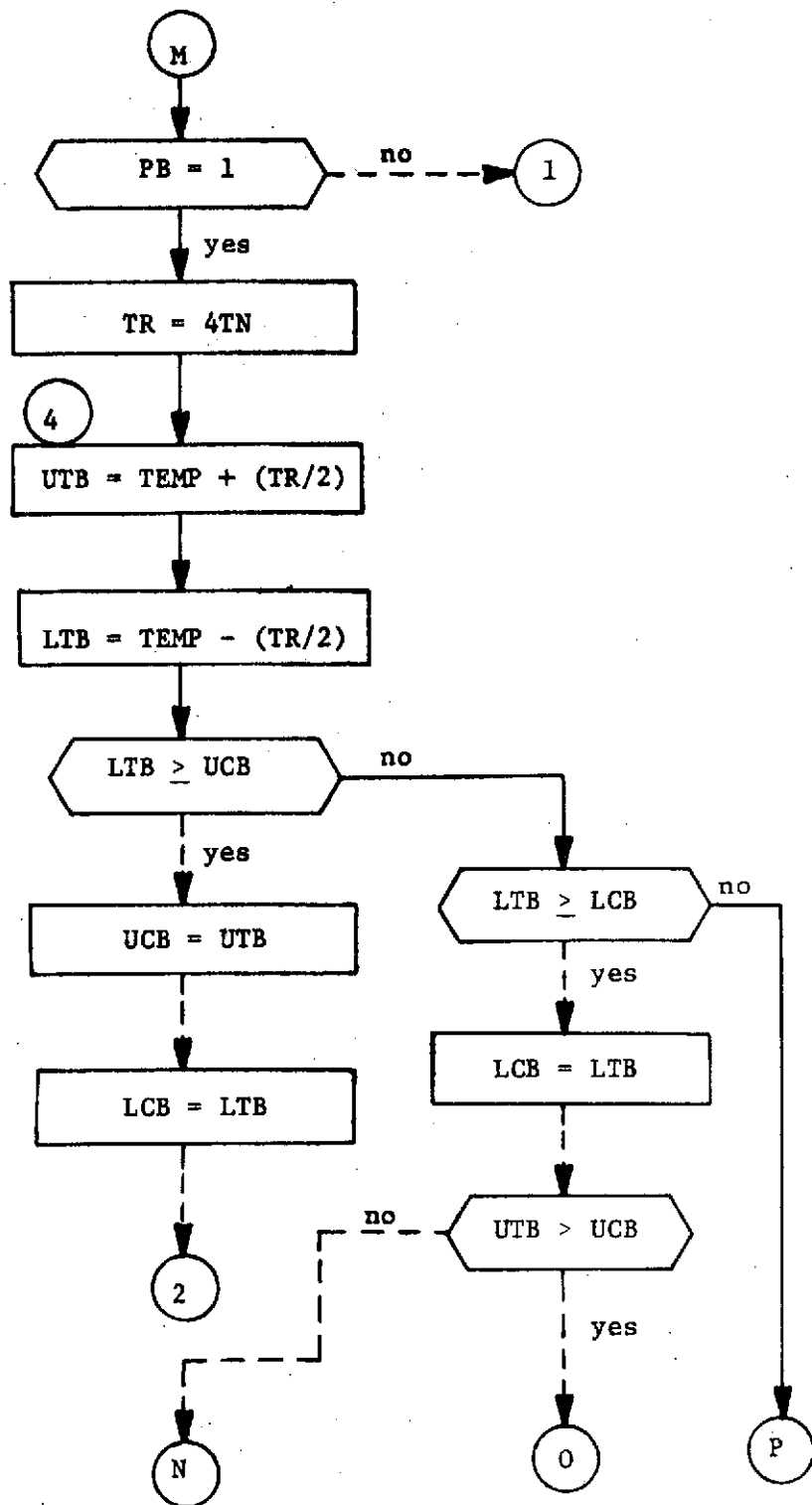


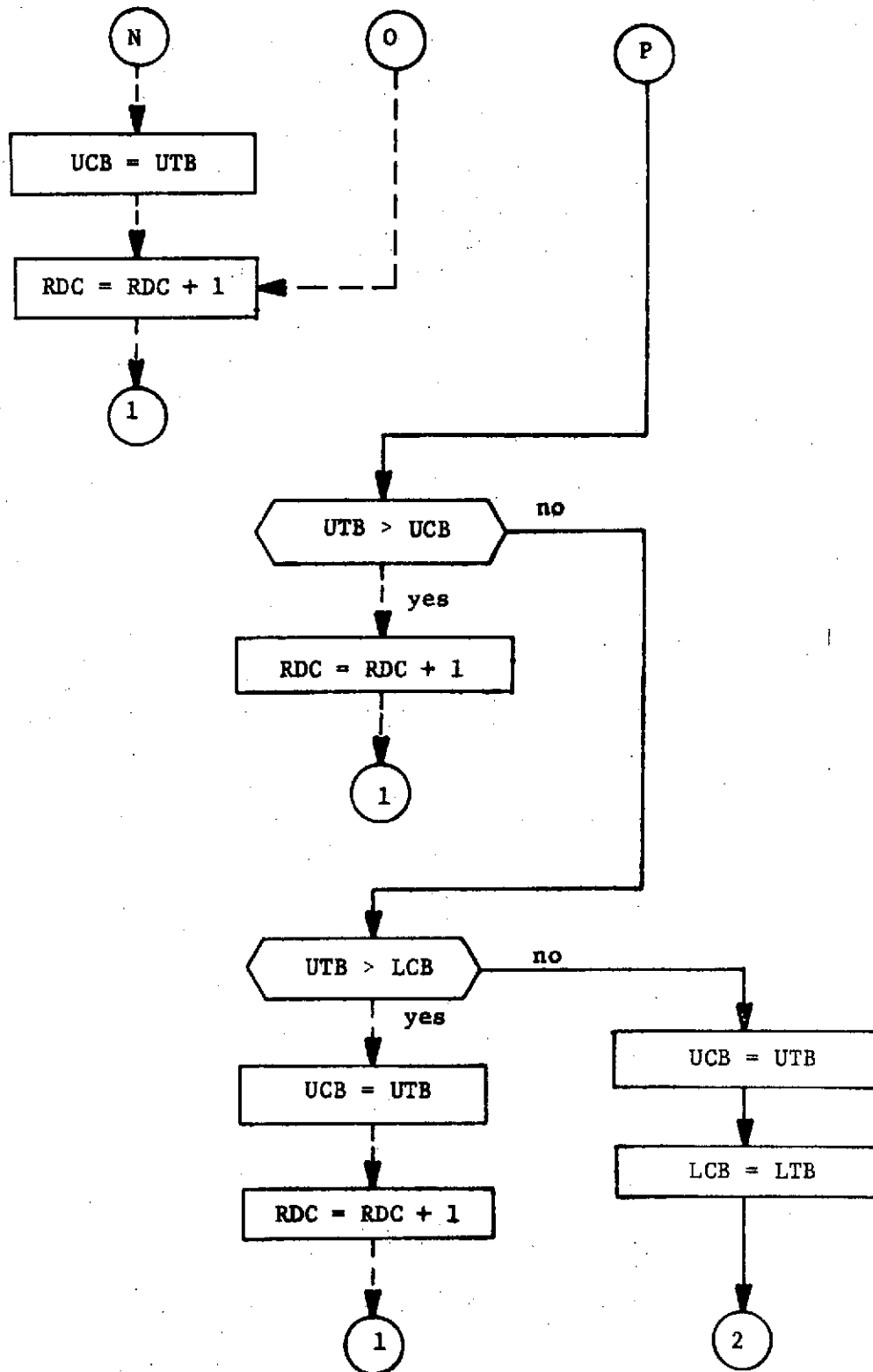












The solid arrows on the flow chart indicate the critical path. Examining the critical path, it is quite easy to see that there are some 59 steps required to complete one iteration, if conditions are such that the critical path is taken. Although an effort was made to simplify the flow chart as much as possible, the fact is immediately obvious that there are still quite a few steps which must be completed if the system is to operate in a manner such as described in Chapter III. In the next section, it will be illustrated how a procedure, whose complexity approaches that of the one presented in the flow chart, will severely limit the speed capabilities of a specific general-purpose computer.

G. The IBM 4 π Model CP General-purpose Digital Computer.

The 4 π -CP computer is the intermediate range model of system 4 π flight computer family. Both data and instructions may be in halfword (16-bit) format or fullword (32-bit) format. Actually, each word is 34 bits long, with the two extra bits being allocated to the functions of parity error detection and storage protection. Since the main concern in this section is the time which will be required to execute the critical path of the data compression system operation as shown in the flow chart, instruction execution times are vitally important. As mentioned in a previous section, most computer manufacturers supply a table of execution times for a given computer instruction set and IBM

is no exception. Table 5-1 shows the table of average execution times for each instruction (both full-word and half-word) in the set for the 4 π -CP computer [11].

In order to obtain some idea of the speed capabilities of the 4 π CP computer, with respect to the desired system operation, the following assumptions will be made.

- 1) The instruction set for the operating program is composed of halfword instructions only. Of course, this would be very difficult to do in practice, because halfword instructions do not have the flexibility of fullword instructions. This assumption, then, is a "best case" assumption, since halfword instructions have the fastest execution time.
- 2) Each instruction will be assumed to require the same amount of execution time as the unindexed halfword LOAD, ADD, and SUBTRACT instructions (3.75 μ sec). Once again this is a "best case" assumption, since these instructions are among those requiring the least amount of execution time. In the actual instruction set which would be dictated by the flow chart, there would be many more instructions which would require more than 3.75 μ sec than instructions which would require less than 3.75 μ sec.
- 3) Each step in the flow chart requires only one instruction execution to complete. This, too, is a "best case" assumption, because many of the steps in the flow chart would require several instructions to complete.

	Halfword		Fullword			
	F = 0		F = 1, IA = 0		F = 1, IA = 1	
	T = 0,1	T = 2,3	T = 0,1	T = 2,3	T = 0,1	T = 2,3
LA	3.75	6.25	5.00	7.50	7.50	10.00
LAH	3.75	6.25	5.00	7.50	7.50	10.00
LQ	3.75	6.25	5.00	7.50	7.50	10.00
SA	4.58	7.08	5.83	8.33	8.33	10.83
1* SAH	4.17	6.67	5.42	7.92	7.92	10.42
SQ	4.58	7.08	5.83	8.33	8.33	10.83
A	3.75	6.25	5.00	7.50	7.50	10.00
AH	3.75	6.25	5.00	7.50	7.50	10.00
2* C	4.79	7.29	6.04	8.54	8.54	11.04
2* CH	3.96	6.46	5.21	7.71	7.71	10.21
AN	3.75	6.25	5.00	7.50	7.50	10.00
OR	3.75	6.25	5.00	7.50	7.50	10.00
XOR	3.75	6.25	5.00	7.50	7.50	10.00
N	18.13	20.63	19.38	21.88	21.88	24.38
D	46.25	48.75	47.50	50.00	50.00	52.50
1* MSH	—	—	5.83	8.33	8.33	10.83
MH	11.46	13.96	12.71	15.21	15.21	17.71
S	3.75	6.25	5.00	7.50	7.50	10.00
SH	3.75	6.25	5.00	7.50	7.50	10.00
1* SXR	4.17	6.67	5.42	7.92	7.92	10.42
LXR	2.08	4.17	3.33	5.63	5.21	7.92
2* MXR	2.08	4.58	3.33	6.05	5.21	8.33
ISPB	—	—	5.42	7.92	7.92	10.42
3* DIOC	4.17	—	—	—	—	—
4* DIOC	2.50	4.38	3.75	5.63	5.63	8.13
5* BSI	5.42	7.92	7.71	10.21	10.21	12.71
6* BSI	—	—	3.75	3.75	3.75	3.75
6* BC	—	—	3.75	3.75	3.75	3.75
5* BC	—	—	5.83	7.50	7.50	10.00
6* BOC	—	—	2.92	2.92	2.92	2.92
5* BOC	—	—	5.00	6.67	6.67	9.17
2* SC	2.71	—	—	—	—	—
SLF	1.88 + 1.25n	3.54 + 1.25n	—	—	—	—
SLD	1.88 + 2.08n	3.54 + 2.08n	—	—	—	—
SLC	1.88 + 1.25n	6.25 + 1.25n	—	—	—	—
SLCD	1.88 + 2.08n	6.25 + 2.08n	—	—	—	—
SRA	1.46 + 0.83n	5.21 + 0.83n	—	—	—	—
SRL	1.46 + 0.83n	5.21 + 0.83n	—	—	—	—
SRAD	3.55 + 1.67n	5.21 + 1.67n	—	—	—	—
7* SRRD	3.55 + 1.67n	5.21 + 1.67n	—	—	—	—
8* SRRD	6.46 + 1.67m	8.13 + 1.67m	—	—	—	—
1* Add 2.5 microseconds if the content of the next instruction location is altered and the next instruction is located on an odd word boundary			5* Branch is executed (includes 1.25 microseconds to read next instruction)			
2* Add 1.25 microseconds if a skip is executed			6* Branch is not executed			
3* F = 0, T = 0; CW = (EA)			7* n < 32			
4* CW = EA			8* When $n \geq 32$, let $m = (n-32)$.			

Table 5-1. Average execution times for the instruction set of the IBM 4π-CP flight computer [11].

Once the above assumptions are made, the calculation of iteration time is quite a simple matter. Recalling that the critical path has some 59 steps which must be performed, it is rather easy to see that the critical path iteration period is

$$(59 \text{ steps}) \times (3.75 \text{ } \mu\text{sec/step}),$$

or

$$221.25 \text{ } \mu\text{sec}.$$

Since this path must be fully executed before another data sample may be accepted, the maximum data rate would be

$$1/221.25 \text{ sec/sample},$$

or

$$\text{roughly } 4.5 \text{ K samples/sec}.$$

This rate, although obtained under "best case" assumptions, is quite unsuitable, because it has already been established that the data rate per channel may reach 120 samples/sec. If there are 64 channels, then the maximum overall data rate might reach 64×120 or 7680 samples/sec.

It is quite possible that the flow chart, and hence the program, might be simplified, thus reducing the number of steps involved. However, it must be stressed again that the figures just obtained were calculated on the basis of "best case" assumptions, and since it was further

established that most of these assumptions would be virtually impossible to realize (with the flow chart presented), one would conclude that it is quite unlikely that a single processor computer with the speed capabilities of the 4 π -CP, would be able to meet the demands which would be imposed.

Some literature, when discussing speed capabilities of general-purpose, obtain an average iteration time based on the execution time required for each path and the probability of execution of each path. To be sure, this method usually produces data rates a good deal higher than those obtained using the critical path method. However, in actual practice, maximum data rate capability is determined by conditions allowing for compression under any conceivable circumstance [8]. Hence, the use of the critical path.

H. Multi-processor General-purpose Computers

Because it has been proposed that the space shuttle be equipped with a parallel processor (3 CPU's), general-purpose computing system, it might seem logical to let this system handle the data compression duties. With the usual high capacity and high speed operation of a multiprocessor system, this solution might, at first, seem quite reasonable. There are, nonetheless, some demands imposed by the operating procedure of the data compression system that might render coexistence with other operating programs impossible.

To begin with, the data compression system function is one of

extremely high priority, i.e. when a data sample is presented to the system, it must be accepted within a period of $1/W$ (see Chapter III) or it will be lost for good. There are other functions, however, which are of the utmost importance, such as guidance. The problem of contention for processing time might best be illustrated by an example.

Consider a tri-processor system onto which various functions, including guidance and data compression, have been programmed. Suppose that for the sake of safety and accuracy, it has been deemed necessary that the guidance program be executed in triple modular redundant (TMR) mode. This being the case, none of the three CPU's may be used for anything else as long as the guidance program is executing. Now suppose that at some time after the guidance program has begun execution a data sample is presented to the system. It is somewhat obvious that if the guidance program does not complete execution within a period of $1/W$, then the data sample will be lost.

By nature, when high priority programs become highly active, it usually indicates that some significant changes are taking place. If this is the case, then the quantity of non-redundant data samples is also going to increase, requiring increased activity on the part of the data compression system. All of this is quite likely to result in a serious contention for processing time and possible indiscriminate loss of important data.

Returning once again to the system operation flow chart, it can be seen that each time a data sample has been processed, the system enters

a "wait" mode until a new data sample arrives. If a general-purpose computer, having sufficient capacity and speed to meet the data compression system requirements, was being used for data compression purposes, then this "wait" time would constitute idle CPU time which could be put to use on other programs. Thus employed, the computer could process low priority programs until an interrupt signaled the arrival of a data sample, at which time the computer would immediately begin processing the data sample. In this manner the general-purpose data compression computer could offer some boost in capacity to the tri-processor computing system. In any case a processor must be assigned to perform data compression as the primary function.

I. Cost

There is very little that can be said about the cost of general-purpose computers that is not already well known fact. Quite naturally, the larger and faster machines are more expensive. Similarly, cost increases in proportion to the amount of main storage that a computer has. Figure 5-1 shows a typical cost spread, for general purpose computer used as data compressors, as a function of maximum sampling rate [5]. This set of curves is based on implementation of the simple data compression algorithms (ZOP, ZOI, FOP, and FOI). One quite astonishing fact that may be observed from Figure 5-1 is that multiple general-purpose computers, that is, multi-processor systems, are less expensive for data compression purposes than are single-processor systems

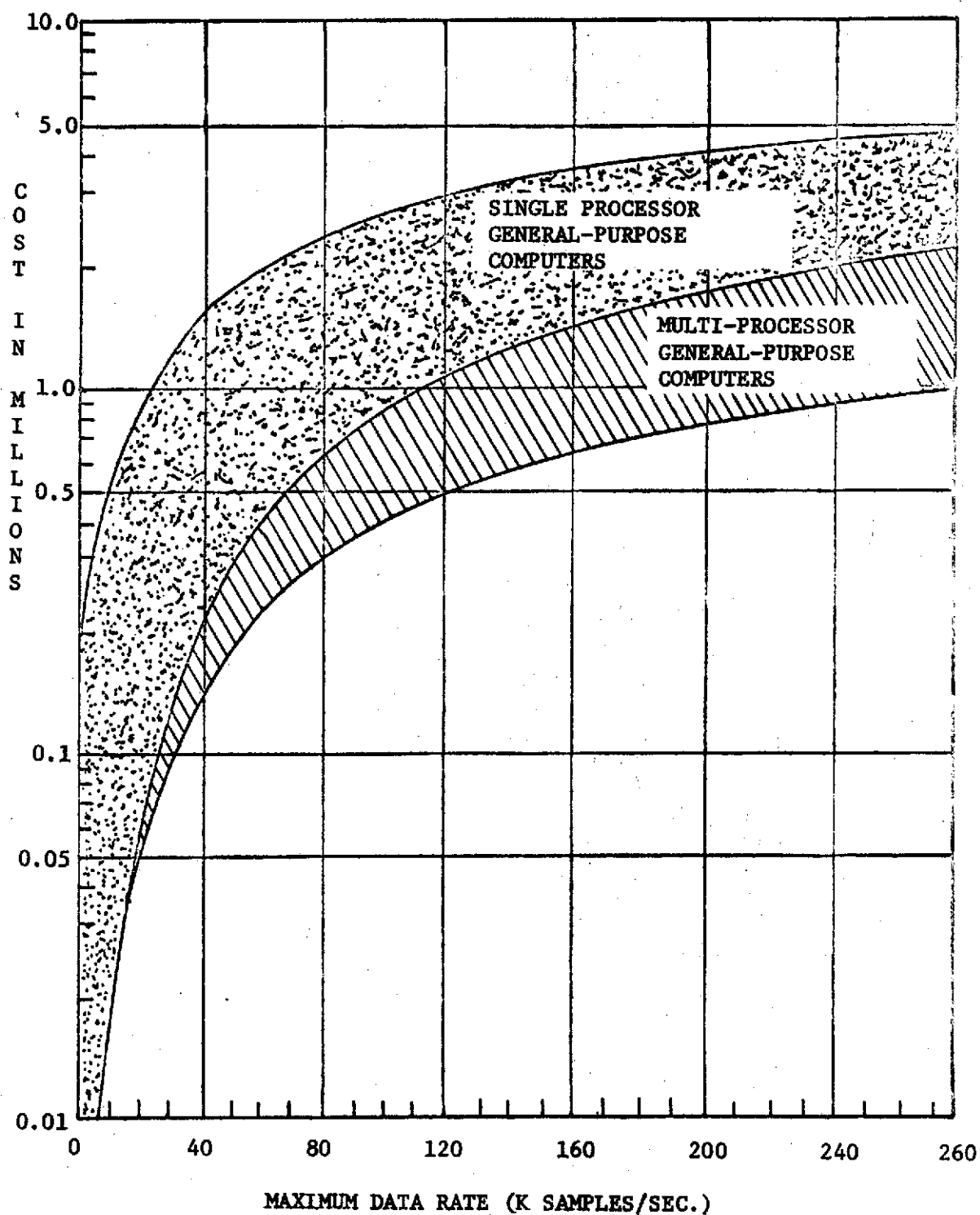


Figure 5-1. Typical cost spread of general-purpose computers used for data compression.

having the same capacity. This is probably due to the fact that slower and less expensive components can be used in the multi-processor systems, while still maintaining the necessary computational capacity.

In summing up, it may be stated that although it is possible to use general-purpose computers for data compression implementation, the demands of the system discussed herein call for a machine with a bit more speed than the 4 π -CP. If it is decided that a general-purpose machine is to be used for data compression, one of the biggest problems might be finding a flight computer with suitable capacity as well as acceptable costs.

VI. Conclusion

From discussion in previous chapters, it is obvious that both special-purpose computers and general-purpose computers can be used for data compression implementation. However, there is a considerable difference in the cost per given performance between these two approaches. Figure 6-1 is a composite of Figures 4-3 and 5-1 and makes a cost comparison between special-purpose and general-purpose data compression implementation somewhat easier [5]. The dotted line on the cost curve shows the approximate minimum yearly lease costs for general-purpose computers. For data input rates of 10K samples/sec. or less, the lower priced general-purpose computers are quite competitive, price-wise, with special-purpose computers. However the lower price range of general-purpose computers would almost certainly not include any type of tactical flight computer with the durability and reliability which would be required by a space mission. Therefore, it must still be concluded that the general-purpose computer will, for the purposes of space borne data compression, be more expensive than its special purpose counterpart.

The future must be considered, also. Suppose that at some time forthcoming, the input data rate is expected to rise to about 40K samples/sec. It would certainly be wise to choose a computer which would be capable of handling this rate, even though the present demands

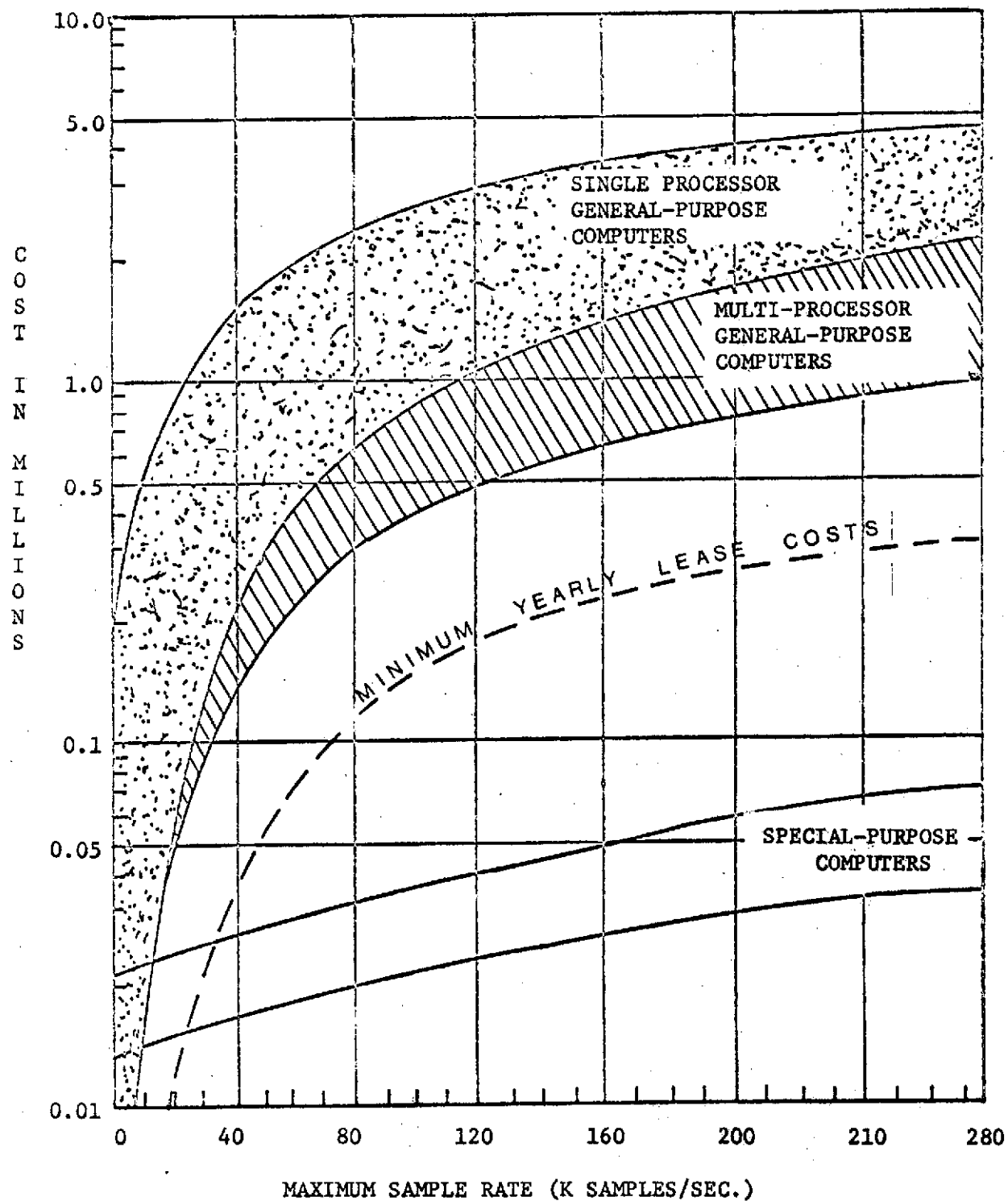


Figure 6-1. Approximate cost spread for general-purpose and special-purpose computers used for data compression.

might not be so strenuous. At 40 K samples/sec., the minimum cost of a single processor, general-purpose computer is about five times the maximum cost of a special-purpose computer with the same capacity. At this input data rate, a multi-processor general-purpose computer might seem like a reasonable alternative to the single processor general-purpose computer, however, it must be remembered that in the space shuttle, there may not be sufficient room available to accomodate an additional multi-processor system. Bearing these facts in mind, it is easy to see that it is economically unsound to use a large, general-purpose computing system when the only function to be performed is that of data compression [7].

Even though the tri-processor, general purpose computer, which is proposed for the space shuttle, may not be fully loaded, it has been pointed out in Chapter V that the coexistence of the data compression system program with other programs which might require the full computing capacity of the system would very possibly lead to indiscriminant loss of important data.

It is because of the facts in this report summary that the contention and recommendation of this report is that a special-purpose computer (either multi-algorithm or single algorithm) be used for data compression implementation aboard the proposed space shuttle craft.

References

- 1) "A Study of Data Systems for the Space Shuttle Vehicle," Technical Report Number 1 prepared by the Digital Systems Laboratory of the Electrical Engineering Department, Auburn University, January 20, 1971.
- 2) Beechtold, W. R. and Smith, W. E., "Preliminary Implementation Studies of Advanced Compression Algorithms," Lockheed Space and Missiles Company, Sunnyvale, California, June 30, 1965.
- 3) "Desired Space Shuttle Data Compression System Operation," prepared by Astrionics Laboratory, Marshall Space Flight Center, Huntsville, Alabama.
- 4) Bruckhelm, A. J., "A Survey of Data Compression Techniques," U. S. Naval Ordinance Laboratory, White Oak, MD., January 12, 1968.
- 5) Weber, D. R., "Some Economic Factors in the Application of Data Compression," Proc. of the National Telemetering Conference, San Francisco, California, 1967.
- 6) Sheldahl, Stephen A., "Comparison of Hardware Requirements for Polynomial Data Compressors," Proc. of the National Telemetering Conference, Houston, Texas, April, 1968.
- 7) Downing, J. J., Smith, W. E., and Stubbles, J. E., "General Purpose Telemetry Data Compression," Electronics Division, Lockheed Missiles and Space Company, Sunnyvale, California, August, 1967.
- 8) Bechtold, W. R., and Hockman, D., "General-Purpose Versus Special-Purpose Computers for Data Compression," Adcom, Inc., Palo Alto, California, August, 1967.
- 9) Frost, W. O., "Considerations in the Design of a Self-Adaptive Data Acquisition System," M. S. Thesis, University of Alabama, Huntsville, 1969.
- 10) Massey, H. N., and Smith, W. E., "Implementation Investigations of ZVA/FVA Compression Algorithms," Summary Report, Lockheed Space and Missiles Company, Sunnyvale, California, February 28, 1966.
- 11) IBM 4 π -CP2 Programmer's Manual, No. 70-928-11, IBM Electronic Systems Center, Owego, New York, October 1, 1970.